

NECパーソナルコンピュータ
PC-9800シリーズ

NEC

PC-9801UX

BASICリファレンスマニュアル



ご注意

- (1) 本書の内容の一部又は全部を無断転載することは禁止されています。
- (2) 本書の内容に関しては将来予告なしに変更することがあります。
- (3) 本書は内容について万全を期して作成いたしましたが、万一御不審な点や誤り、記載もれなどお気づきのことがありましたら御連絡下さい。
- (4) 運用した結果の影響について(3)項にかかわらず責任を負いかねますので御了承下さい。

© 1986, 1987 NEC Corporation

Original Copyright © 1981, 1987 ASCII Techwrite

輸出する際の注意事項

本製品（ソフトウェアを含む）は、外国為替および外国貿易管理法の規定により、戦略物資等輸出規制品に該当します。従って、日本国外に持出す際には日本国政府の輸出許可申請等必要な手続きをお取り下さい。

日本電気株式会社の許可なく複製・改変などを行うことはできません。

BASICリファレンスマニュアル

はじめに

PC-9800 シリーズパーソナルコンピュータには標準プログラミング言語としてN₈₈-BASIC(86)が用意されています。

N₈₈-BASIC(86)言語で記述されたプログラムは、本体ROMあるいは各種インタフェースボードに搭載されているN₈₈-BASIC(86)インタプリタにより解釈／実行されます。

また、N₈₈-日本語BASIC(86)システムディスクを使用することにより、ROMに搭載されているN₈₈-BASIC(86)インタプリタの持つ機能に加え、さまざまな機能を利用することができます。

本書は、N₈₈-BASIC(86)言語が持つ機能を、命令・関数単位で詳細に解説した文法説明書です。

N₈₈-BASIC(86)インタプリタの操作法、動作環境、あるいは応用的な操作(日本語入力など)については、「BASICユーザーズマニュアル」を参照してください。

N₈₈-BASIC(86)言語の各命令・関数の機能を簡略に説明してある「BASICリファレンスブック」は、実際のプログラミング時などにハンドブックとしてお使いください。

具体的なプログラミングの方法については「BASICプログラミング入門」をご覧ください。

このマニュアルの構成

本書は、次の4章および付録から構成されています。

第1章 N₈₈-BASIC(86)について

N₈₈-BASIC(86)の特徴、主な機能のほか、ROMモードBASICとDISKモードBASICの違いや、プログラムの作り方などの基本的な事項を解説しています。

第2章 BASICの文法

N₈₈-BASIC(86)言語の文法・規則についての解説です。本BASICを利用する前には必ず本章を読み、その内容を理解・把握しておいてください。

第3章 命令リファレンス

N₈₈-BASIC(86)で使用可能な命令・関数をアルファベット順に並べ、個々に詳細な解説をしています。本BASICの詳しい辞書として利用してください。

第4章 サンプルプログラム

相互に関連のある命令・関数を使用して作成したサンプルプログラム集です。実際にプログラムを作成する際の参考にしてください。

付 録

エラーメッセージとその対策、キャラクタコード表、予約語一覧などの付加情報・資料です。

目 次

はじめに	(3)
このマニュアルの構成	(4)
命令・関数の機能別索引	(10)

第 1 章 N₈₈-BASIC (86) について

1. ROMモード BASICとDISKモード BASIC	3
2. N ₈₈ -BASIC (86) の主な機能	3
3. BASICの動作モード	4
3.1 ダイレクトモード	4
3.2 プログラムモード	5
・プログラムの作り方	
・プログラムの修正	
・プログラムの保存(セーブ)と読み込み(ロード)	

第 2 章 BASICの文法

1. 文	9
2. 行	9
3. 行番号	9
4. BASICで使える文字と特殊記号	10
・特殊記号の使い方	
5. 定数	12
5.1 文字型定数	12
5.2 数値型定数	12
5.3 整数型定数	12
・10 進形式	
・8 進形式	
・16 進形式	
5.4 実数型定数	13
5.5 単精度実数型定数	13
5.6 倍精度実数型定数	14
6. 変数	14
6.1 変数名	14

6.2 変数の型	14
7. 配列変数	15
8. 予約語	17
9. 型変換	17
10. 式と演算	19
10.1 算術演算	19
• 整数の除算と剰余の計算	
• 0 での除算	
• 桁あふれ(オーバーフロー)	
10.2 関係演算	21
10.3 論理演算	21
10.4 関数	24
10.5 文字列の演算	24
• 文字列の連結	
• 文字列の比較	
10.6 演算の優先順位	25
11. ファイル	25
11.1 ファイルとは	25
11.2 ファイルディスクリプタ	26
• デバイス名	
• ファイル名	
11.3 データファイルとファイル番号	27
12. 割り込み	28
13. ラベル名	29
14. エラーメッセージ	30

第 3 章 命令リファレンス

この章の見方	35
ABS	38
AKCNV\$	38
ASC	38
ATN	39
ATTR\$	39
AUTO	40
BEEP	40
BLOAD	40
BSAVE	41
CALL	42
CDBL	42
CHAIN	43
CHR\$	44
CINT	44

CIRCLE	45	FIELD	72
CLEAR	46	FILES/LFILES	73
CLOSE	47	FIX	74
CLS	47	FOR...TO...STEP~NEXT	74
[1] COLOR	48	FPOS	75
[2] COLOR	51	FRE	75
COLOR@	52	GET	76
COMMON	52	GET@	77
COM ON/OFF/STOP	53	GOSUB	78
CONSOLE	54	GOTO/GO TO	78
CONT	55	HELP ON/OFF/STOP	79
COPY	55	HEX\$.....	80
COS	56	IF...THEN~ELSE/IF...GOTO	
CSNG	57	~ELSE	80
CSRLIN	57	INKEY\$	81
CVI/CVS/CVD	58	INP	81
DATA	58	INPUT	82
DATE\$	59	INPUT#	83
DEF FN	59	INPUT\$	83
DEFINT/DEFSNG/DEFDBL/		INPUT WAIT	84
DEFSTR	60	INSTR	84
DEF SEG	61	INT	85
DEF USR	61	JIS\$	85
DELETE	62	KACNV\$	85
DIM	62	KEXT\$	86
DRAW	63	KEY	86
DSKF	67	KEY LIST	87
DSKI\$	68	KEY ON/OFF/STOP	87
DSKO\$	68	KILL	88
EDIT	69	KINPUT	88
END	70	KINSTR	89
EOF	70	KLEN	89
ERASE	70	KMID\$	90
ERL/ERR	71	KNJ\$	90
ERROR	71	KPLOAD	91
EXP	72	KTYPE	92

LEFT\$	92	ON TIME\$ GOSUB	113
LEN	93	OPEN	114
LET	93	OPTION BASE	116
LINE	94	OUT	117
LINE INPUT	95	[1] PAINT	117
LINE INPUT#	96	[2] PAINT	118
LINE INPUT WAIT	96	PEEK	120
LIST/LLIST	97	PEN	120
LOAD	97	PEN ON/OFF/STOP	121
LOAD?	98	POINT	122
LOC	98	[1] POINT	122
LOCATE	99	[2] POINT	123
LOF	99	POKE	123
LOG	100	POS	124
LPOS	100	PRESET	124
LPRINT	100	PRINT	125
LPRINT USING	101	PRINT#	126
LSET/RSET	101	PRINT USING	128
MAP	102	PRINT# USING	130
MERGE	103	PSET	130
MID\$	103	PUT	131
MID\$(関数)	104	PUT@	132
MKIS/MKS\$/MKD\$	104	RANDOMIZE	133
MON	105	READ	134
MOTOR	105	REM	134
NAME	106	RENUM	135
NEW	106	RESTORE	135
NEW ON	106	RESUME	136
OCT\$	107	RETURN	136
ON COM GOSUB	108	RIGHT\$	137
ON ERROR GOTO	109	RND	137
ON...GOSUB/ON...GOTO	109	ROLL	138
ON HELP GOSUB	110	RUN	138
ON KEY GOSUB	111	SAVE	139
ON PEN GOSUB	112	SCREEN	140
ON STOP GOSUB	112	SEARCH	143

SET	144	TIME\$ ON/OFF/STOP	152
SGN	145	TRON/TROFF	153
SIN	145	USR	153
SPACE\$	146	VAL	154
SPC	146	VARPTR	154
SQR	146	VIEW	155
STOP	147	VIEW(関数)	156
STOP ON/OFF/STOP	147	WAIT	157
STR\$	148	WHILE~WEND	158
STRING\$	148	WIDTH	158
SWAP	149	WIDTH LPRINT	159
TAB	149	WINDOW	160
TAN	149	WINDOW(関数)	161
TERM	150	WRITE	161
TIME\$	152	WRITE#	162

第4章 サンプルプログラム

この章の見方	165
サンプルプログラム	168

付 録

A. エラーメッセージとその対策	181
B. コントロールコード表	189
C. キャラクタコード表	190
D. 誘導関数	191
E. 予約語一覧	192

索 引	195
-----------	-----

命令・関数の機能別索引

コマンド

プログラム編集用命令

AUTO	40
DELETE	62
EDIT	69
KEY LIST	87
LIST/LLIST	97
LOAD	97
MERGE	103
RENUM	135
SAVE	139

プログラム実行制御用命令

NEW	106
NEW ON	106
RUN	138
TRON/TROFF	153

ファイル操作命令

FILES/LFILES	73
KILL	88
NAME	106
SET	144

一般命令

命 令

CHAIN	43
COMMON	52
DATA	58
DEF FN	59
DEFINT/DEFSNG/DEFDBL/DEFSTR	60
DIM	62
END	70
ERASE	70
FOR...TO...STEP~NEXT	74
GOSUB	78
GOTO/GO TO	78
IF...THEN~ELSE/ IF...GOTO~ELSE	80

LET	93
ON...GOSUB/ON...GOTO	109
OPTION BASE	116
RANDOMIZE	133
READ	134
REM	134
RESTORE	135
RETURN	136
STOP	147
SWAP	149
WHILE~WEND	158

関 数

SEARCH	143
--------------	-----

— テキスト画面制御 —

命 令

CLS	47
[1]COLOR	48
COLOR@	52
CONSOLE	54
LOCATE	99
PRINT	125
PRINT USING	128

WRITE	161
-------------	-----

関 数

CSRLIN	57
POS	124
SPC	146
TAB	149

— グラフィック画面制御 —

命 令

CIRCLE	45
CLS	47
[2]COLOR	51
DRAW	63
GET@	77
LINE	94
[1]PAINT	117
[2]PAINT	118
POINT	122
PRESET	124
PSET	130

PUT@	132
ROLL	138
SCREEN	140
VIEW	155
WINDOW	160

関 数

MAP	102
[1]POINT	122
[2]POINT	123
VIEW	155
WINDOW	160

算術関数

関 数	
ABS	38
ATN	39
CDBL	42
CINT	44
COS	56
CSNG	57
CVI/CVS/CVD	58
EXP	72
FIX	74
INT	85
LOG	100
RND	137
SGN	145
SIN	145
SQR	146
TAN	149

エラー制御

命 令		関 数	
ERROR	71	ERL/ERR	71
ON ERROR GOTO	109		
RESUME	136		

1 バイト系文字列操作

命 令		関 数	
MID\$	103	LEN	93
		MID\$	104
		MKI\$/MKS\$/MKD\$	104
		OCT\$	107
		RIGHT\$	137
		SPACE\$	146
		STR\$	148
		STRING\$	148
		VAL	154
ASC	38		
CHR\$	44		
HEX\$	80		
INSTR	84		
LEFT\$	92		

日本語文字列操作

命 令

KPLOAD91

関 数

AKCNV\$38

JIS\$85

KACNV\$85

KEXT\$86

KINSTR89

KLEN89

KMID\$90

KNJ\$90

KTYPE92

ファイル制御

命 令

CLOSE47

FIELD72

GET76

INPUT#83

LINE INPUT#96

LSET/RSET101

OPEN114

PRINT#126

PRINT# USING130

PUT131

WRITE#162

関 数

ATTR\$39

DSKF67

EOF70

FPOS75

INPUT\$83

LOC98

LOF99

キー制御

命 令			
HELP ON/OFF/STOP	79	LINE INPUT WAIT.....	96
INPUT	82	ON HELP GOSUB	110
INPUT WAIT	84	ON KEY GOSUB	111
KEY.....	86	ON STOP GOSUB	112
KEY ON/OFF/STOP	87	STOP ON/OFF/STOP	147
KINPUT	88		
LINE INPUT.....	95	関 数	
		INKEY\$	81

プリンタ制御

命 令		関 数	
COPY	55	LPOS	100
LPRINT	100	SPACE\$	146
LPRINT USING	101	SPC	146
WIDTH LPRINT	159		

時刻／日付け制御

命 令		関 数	
ON TIME\$ GOSUB	113	DATE\$	59
TIME\$ ON/OFF/STOP.....	152	TIME\$	152

RS-232C コミュニケーションポート制御

命 令			
COM ON/OFF/STOP	53	ON COM GOSUB	108
		OPEN	114

ライトペン制御

命 令	関 数
ON PEN GOSUB112	PEN120
PEN ON/OFF/STOP.....121	

I/O ポート入出力制御

命 令	関 数
BEEP40	INP81
OUT117	
WAIT157	

カセットテープ制御

命 令	関 数
LOAD?98	MOTOR105

メモリ管理

命 令	関 数
CLEAR46	FRE75
DEF SEG61	PEEK120
POKE123	VARPTR154

機械語プログラム制御

命 令	関 数
BLOAD40	DEF USR61
BSAVE41	
CALL42	USR153

モード変更	
命 令	
MON	105
TERM	150

ディスク入出力	
命 令	関 数
DSKO\$	68
DSKI\$	68

第1章

1

N₈₈-BASIC(86)について

第1章 N₈₈-BASIC(86)について

1. ROM モード BASIC と DISK モード BASIC

PC-9800 シリーズパーソナルコンピュータには、標準プログラム言語として N₈₈-BASIC (86)が用意されています。

N₈₈-BASIC (86) には、DISK モード BASIC と ROM モード BASIC の2つのモードがあります。

ROM モード BASIC は、コンピュータ本体の ROM に組み込まれている基本機能だけを使ったシステムです。システムディスクを使用しないで、電源を ON にすると起動します。

DISK モード BASIC は、ROM モード BASIC に、ディスクファイルに対する入出力や逐次変換による日本語入力などの機能を拡張したものです。DISK モード BASIC を起動するには、本体の電源を ON にした後、N₈₈-日本語 BASIC (86) システムディスクをフロッピーディスク装置にセットしてリセットスイッチを押します。

ROM モード BASIC でもプログラムの入力および実行は可能ですが、実際のプログラミングには DISK モード BASIC が適しています。

注) BASIC の起動方法について詳しくは、BASIC ユーザーズマニュアルを参照してください。

2. N₈₈-BASIC(86)の主な機能

N₈₈-BASIC(86)には、次のような特長があります。

(1) グラフィック機能の充実

- 最大640×400の高分解能、最高4096色中16色(DISK モードのみ)のカラー表現によるグラフィックス。
- 強力なグラフィック命令。
- テキスト画面とグラフィック画面の合成。
- パレットの導入による多彩なカラー表示(DISK モードのみ)。

(2) プログラミングのしやすさ

- プログラム中の飛び先指定としてラベル名の使用が可能。
- BASIC 実行時でも、大容量のユーザーメモリを確保しているため、大規模なプログラムの作成が可能。
- 各種ハードウェアからの割り込みを BASIC 中で操作可能。

- IF…THEN～ELSE や, WHILE～WEND などより, 構造化されたプログラムの作成が可能.
 - CHAIN によるプログラム連結機能 (DISK モードのみ).
 - 変数名として, 最長40文字まで使用可能.
- (3) フロッピーディスク, 固定ディスクの使用が可能 (DISK モードのみ).
- (4) 逐次/連文節変換あるいは単文節変換による日本語入力の機能 (DISK モードのみ).
- (5) モデム-NCU 内蔵電話機の制御機能 (DISK モードのみ).
- (6) 倍精度数値関数のサポート (DISK モードのみ).

3. BASIC の動作モード

BASIC を起動すると, 画面には,

How many files (0-15) ?

という文字が表示されます. このメッセージは “同時にいくつのファイルを使用しますか” と尋ねているものです. ここで数を指定すると, その数だけのファイルを同時に開くことができるようになります. 詳しくは, 第3章中の OPEN の項を参照してください.


ここで, 数を指定してからリターンキーを押すか, またはたんにリターンキーを押すと, BASIC のバージョンなどを示すメッセージが表示されます. メッセージ中最後行の “Ok” という文字は BASIC の命令をキーボードから入力することができることを示しているもので, “プロンプト” と呼ばれます.

BASIC の命令は, ダイレクトモードとプログラムモードの2つの動作モードで実行されます.

3.1 ダイレクトモード

BASIC の命令を, 単独で実行させるモードをダイレクトモードといいます. ダイレクトモードでの命令の実行は, 命令をキーボードから打ち込み, リターンキーを押すという動作 (これを命令の入力といいます) によって行われます.

たとえば, 「PRINT 21+6-5」とキーを打ち込み, 終わりにリターンキーを押してみてください.

PRINT 21+6-5 

22

Ok

となります。PRINT は“画面にデータを出力せよ”という命令ですから、この場合には、“21+6-5”の計算結果として“22”が表示されます。

このように、ダイレクトモードで BASIC の命令を実行するときには、命令の最後にリターンキーを押さなければなりません。リターンキーを押すことによって、初めて命令がコンピュータに送られるのです。

ダイレクトモードで入力された命令はコンピュータのメモリに格納されませんので、その場限りの計算や記憶する必要のない命令を実行するのに使います。

なお、ダイレクトモードで、マルチステートメント（「第2章 2. 行」参照）を使用してもかまいません。

3.2 プログラムモード

BASIC の命令を続けて実行させるためには、プログラムを作る必要があります。プログラムを入力し、実行するモードをプログラムモードといいます。

■プログラムの作り方

プログラムを作るときには、命令の先頭に数字をつけて入力していきます。各行の終わりに必ずリターンキーを押します。すると、行は数字とともにメモリに格納されます。この数字を行番号といいます。

プログラムを実行するには、プログラムを入力した後で、RUN を入力します。プログラムは行番号の小さいものから順に実行されます。

```
10 A=2
20 B=-5
30 PRINT A+B
RUN
-3
Ok
```

メモリにはいちどに1つのプログラムしか格納することはできません。メモリに古いプログラムがあるときは、新しいプログラムを入力する前に、必ず NEW を実行して、古いプログラムを消しておかねばなりません。

なお、プログラムモードでは、マルチステートメント（「第2章 2. 行」参照）を利用することができます。

■プログラムの修正

(1) 行の追加

プログラムは行番号の小さいものから実行されます。したがって、新しく行を追加したい場合は、追加したい所の行番号(前後の行番号の間の番号)をつけた命令を入力します。た

たとえば前記のプログラムの20行と30行の間に命令を入れるときには21～29までの行番号をつけることができます。

(2) 行の削除

削除したい行の行番号だけを打ち込んでリターンキーを押せば、その行は削除されます。まとめて削除するときには、DELETE を使います (第3章中の DELETE の項参照)。

(3) 行の置き換え

訂正したい行の行番号と新しい内容を打ち込んでリターンキーを押せば、古い行の内容は自動的に新しいものになります。また、LIST を使ってプログラムを画面に表示し、カーソルを直したい所に移動して直接書き換えてからリターンキーを押しても、その行は訂正されます (第3章中の LIST の項参照)。

(4) 行番号のつけかえ

行番号をつけかえたいときには、RENUM を使います (第3章中の RENUM の項参照)。

■プログラムの保存(セーブ)と読み込み(ロード)

BASIC で作成したプログラムは、本体の電源を切ると消えてしまいます。そのため、必要なプログラムは、ファイルという形でディスクに書き出し、保存(セーブ)しておかなければなりません。

プログラムをセーブするには、SAVE を用います。プログラムは一度セーブしてしまえば、あとはいつでもメモリに読み込み(ロード)、使うことができます。プログラムをロードするには、LOAD を使います。

機械語プログラムをセーブ、ロードするための、BSAVE と BLOAD も用意されています。詳しくは第3章の各命令の項を参照してください。

なお、ディスクに対するセーブ・ロードは、DISK モードでのみ可能です。また、別売の CMT インタフェースボードを使用すれば、カセットテープに対してセーブ・ロードを行うこともできます。

第2章

2

BASICの文法

第2章 BASICの文法

1. 文

文は、BASIC が行う手続き（すなわちコンピュータに実行させるべき処理）を記述する最小単位です。

文は1つの命令からなります。そして命令は、命令の名前と、命令に与える情報（パラメータ）とからなります。パラメータには定数、変数、関数（「第2章 10.4 関数」参照）のほか、これらを組み合わせた式などを指定することができます。

<u>P R I N T</u>	<u>1 2 3 4</u>
命令の名前	パラメータ
命令／文	

2. 行

行とは、プログラムモードで実行する場合において、メモリ中に格納（記憶）されるプログラムとして書かれる、行単位の文の集まりのことです。

行の先頭には行番号をつけます。1行（行番号も含めて）には255バイトの範囲で文の記述が可能です。

行は、基本的には1つの文からなりますが、複数の文を記述することも可能です。1行に複数の文を記述する場合、各文をコロン（:）で区切ります。これは複文（マルチステートメント）と呼ばれ、1行の範囲内で任意の数の複文が記述できます。

3. 行番号

行番号は1から65529までの整数で指定します。

行番号は行をメモリに格納したりディスクにセーブしたりする場合の格納順序を示すものです。

行番号をつけた行は、リターンキーの入力とともにメモリに格納されます。この方法で1行以上の行をメモリに格納したものがプログラムであり、RUN などを使って実行させることができます。

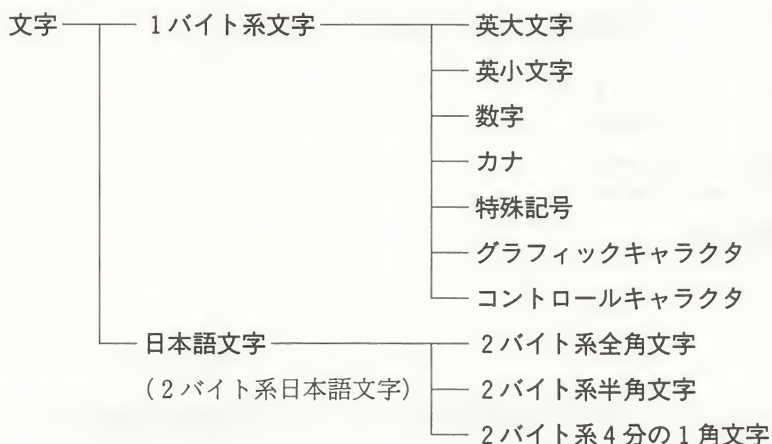
このように、行番号をともなった行を入力する方法が“プログラムモード”です。これに対し、行番号なしで命令を入力することによってただちに実行を行う方法が“ダイレクトモード”です（「第1章 3. BASIC の動作モード」参照）。

プログラムの実行は行番号の小さいものから行われます。

行番号は分岐（プログラムの実行順序を変えること）の場合の分岐先として使用されるほか、LIST、DELETE など〈行番号〉を対象とする命令のパラメータ（引数）として使われます。

4. BASIC で使用できる文字と特殊記号

使用できる文字は次のとおりです。



1 バイト系の英大文字と英小文字は、キーボード上から入力できる通常の英文字です。BASIC では、特定の場所（たとえば文字型定数の中…文字型定数に関しては「第2章 5.1 文字型定数」参照）を除き、英大文字と小文字とは同じ意味に解釈されます。

コントロールキャラクタは、スクリーンエディタに直接指示を与えるために使うもので、キーボード上の **CTRL** キーを押しながら英文字 1 文字のキーを押すことによって入力します。

日本語文字は、日本語処理システムによって入力する文字です。命令によっては使えないものもありますので注意してください。

これらの文字の詳細については、「付録B. コントロールコード表」、「付録C. キャラクタコード表」、あるいは BASIC ユーザーズマニュアルの「キャラクタコード表」、「日本語コード表」、「第5章 日本語入力」、「第6章 日本語処理」、ハードウェアマニュアルの「漢字コード表」などを参照してください。

■特殊記号の使い方

算術演算子（＋，－，＊，／）などの他にも特別な意味を持つ記号があります。ここでその意味をまとめて説明しておきます。なお、ここで説明しているのは 1 バイト系の特殊記号です。

2 バイト系日本語文字の中にも同様な記号がありますが、それらは文字型定数としてのみ使えるもので、特殊記号として使うことはできません。

(1) ピリオド (.)

最後に BASIC の処理の対象となった行 (現在行, 着目行などという) の行番号の値を示す記号で、命令中の〈行番号〉の代わりとして使用することができます。たとえば、新しい行を挿入した、エラーが発生したなどの行です。

例) LIST .

(2) マイナス (-)

LIST, DELETE などで行の範囲を指定するとき、何行から何行までという場合に使います。算術演算子の負号と同じ記号です。

例) DELETE 100-200

(3) コロン (:)

マルチステートメントの区切りとして使います。

例) A=B+C : PRINT A

(4) コンマ (,)

パラメータが並ぶ場合その区切りとして使います。

例) INPUT A, B, C
COLOR 7, , 0

(5) セミコロン (;)

PRINT などの出力文中でパラメータが並ぶ場合その区切りとして使います。

例) PRINT "A=" ; A

(6) アポストロフィ (')

REM (第3章中の REM の項参照) の代用として使います。

(7) クエッションマーク (?)

PRINT の代用として使います。

(8) アスタリスク (*)

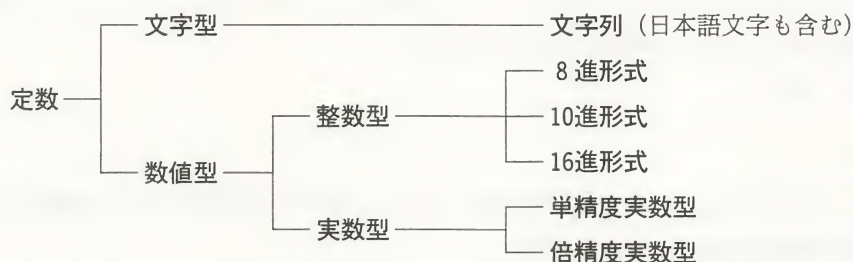
ラベル名の先頭につけます (「第2章 13. ラベル名」参照)。

(9) スペース

原則として文字型定数に含まれているスペース以外のスペースは無視されますが、命令の名前の直後には必ずスペースを入れなければなりません。

5. 定数

定数とは、それぞれ固有の値を持ったデータのことで、プログラム中などに直接表記されるものです。定数は、次のように分類することができます。定数の表記法は、型によってそれぞれ異なります。



5.1 文字型定数

文字型定数とは、255バイト以内の文字をつなぎ、その前後をダブルクォーテーション (") で囲んだ、文字列データのことで、文字列中には、1バイト系の英数字、カナ文字、特殊記号、グラフィックキャラクタならびに2バイト系日本語文字などを使用できます（混在も可能です）。

BASIC の命令のなかには、2バイト系日本語文字が使用できないものもあります（各命令参照）。また、(") を文字型定数中に直接記述することはできません。(") を文字型定数としたい場合、CHR\$を用いて、例のように CHR\$(&H22)とします。

なお、文字型定数を算術演算に使うことはできません。

例) "Good Morning"

"1 2 3 4 5 6 7 8 9"

"パーソナル コンピュータ"

"日本語 BASIC"

CHR\$(&H22)+ "TEST"+CHR\$(&H22)

5.2 数値型定数

数値型定数とは、算術演算を行うことのできる数値データを直接表記したもので、大きく分けて、整数型と実数型とがあります。

5.3 整数型定数

整数型定数は、表記上の観点から、8進・10進・16進の3つの形式に分類されます。

■10進形式

−32768から+32767までのすべての整数です。負の整数の場合、数値の前には必ずマイナス

符号をつけなければなりません。正の数の前の符号は省略できます。また、同じ範囲内の実数の後ろに%をつけると、小数点以下は四捨五入され、整数型の定数とみなされます。

例) 32767
 -123
 +5
 31100.5% ←整数を表す。

■ 8進形式

頭に&O または&をともなった、0 から 7 までの数字の並び、&0～&177777の範囲内です。

例) &12345
 &07777

■ 16進形式

頭に&H をともなった、0 から F までの並び、&H0～&HFFFF の範囲内です。

例) &H100
 &HCFFF

注意：8進形式または16進形式で入力された数値は、PRINT などの出力文では10進形式で出力されます。10進以外の形式で出力するときは、それぞれ OCT \$, HEX \$ を使って文字列として出力してください。

5.4 実数型定数

実数型定数は、単精度実数型と倍精度実数型に分けられます。

5.5 単精度実数型定数

有効桁 7 桁の精度をもつ実数のデータです。出力のときは 7 桁目が四捨五入され、6 桁以下で表示されます。単精度型実数の値の範囲は、 $-1.70141\text{E}+38$ ～ $1.70141\text{E}+38$ です。

BASIC では、とくに型宣言をしない場合、数値はみな単精度実数型として扱われます。

単精度実数型の定数には、表記上の分類により、次の 3 つの形式があります。

■ 7 桁以下の実数

■ 最後に ! をともなった数

■ E を使った指数形式

例) 3525.68
 3.14 !
 -7.09E-06

5.6 倍精度実数型定数

有効桁16桁の精度をもつ実数のデータです。出力のときは、16桁以下で表示されます。倍精度型実数の値の範囲は、 $-1.701411834604692D+38 \sim 1.701411834604692D+38$ です。

倍精度実数型の定数には、表記上の分類により、次の3つの形式があります。

■ 8桁以上の実数

■ 最後に#をとめた数

■ Dを使った指数形式

例) 1234567.890
56789.0#
-1.09432D-38

6. 変数

変数は、プログラム中で使うデータをしまっておくことができる入れ物のようなもので、1バイト系の英数字で名前(変数名)をつけます。変数は、演算、参照などに使うことができます。

なお、変数に値を格納する(入れる)前は、数値変数の値は0、文字変数はヌルストリング(空の文字列)であるものとみなされます。

6.1 変数名

変数名は長さ40文字以内の、1バイト系英数字とピリオド(.)の組み合わせで表されます。ただし、変数名の初めの1文字は英字でなくてはなりません。この規約にしたがって名前をつけられた変数は、すべて区別されます。

たとえば、次の2つの変数は異なった変数名として解釈されます。

```
COUNTER.OF.TABLE.DATA.999888777666555.01
```

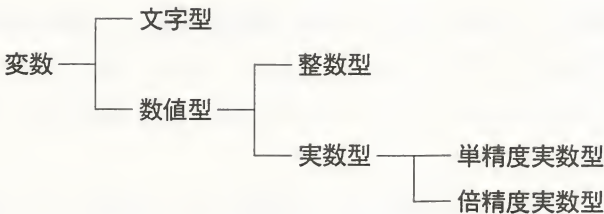
```
COUNTER.OF.TABLE.DATA.999888777666555.02
```

変数名は予約語(「第2章 8. 予約語」, 「付録E. 予約語一覧」参照)であってはなりませんが、予約語を含んだものはかまいません。ただし、FNで始まる変数名は許されません(第3章中のDEF FNの項参照)。また英文字において大文字、小文字の区別はありません。

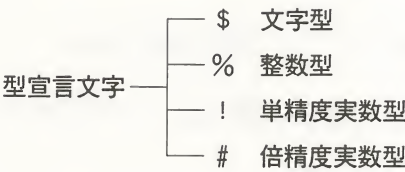
6.2 変数の型

変数にも、定数と同様、格納するデータに応じた型があります。

変数に値を格納するためには、代入文やINPUTなどを用いますが、いずれの場合にも変数の型は、格納されるデータの型と一致していなければなりません。



変数の型は、変数名の最後に型宣言文字をつけることによって区別します。型宣言文字の種類は次の 4 つです。



通常は、型宣言文字を省略すると、“!”がついているとみなされます（単精度実数型変数となる）。

例)

A !	}	これらは区別されるが、A ! と A は同じ。
A #		
A %		
A \$		

ただし、型宣言命令 (DEFINT, DEFSNG など) を用いて一括指定を行った場合、型宣言文字省略時の型は、型宣言命令によって宣言された型となります。たとえば、次の例では、A～C で始まる変数は、型宣言文字をつけないとみな整数型となります。

例)

```
DEFINT A-C
{
  C=A+B    ← A, B, C は、型宣言文字をつけないければ整数型
  L=M*N    ← A～C 以外は、型宣言文字をつけないければ単精度実数型
```

7. 配列変数

配列変数は、たくさんのデータを変数に代入したり、参照する場合に使います。たとえば、データの数 が 100 個あるとき、100 個の別の変数を用意するのはたいへんです。このようなときに、A (0) ～A (99) のように変数名の後にカッコつきの番号をつけた形で配列として扱くと、100 個のデータを 1 つの変数名で処理することができます。

カッコ内の番号は添字といい、配列のなかの何番目の要素であるかを表します。たとえば、A (0) は、A という配列変数の 0 番目の要素という意味になります。添字には整数定数、または整数型の変数を使うことができます。

配列変数の型(すなわち配列変数内の各要素の型)は、通常の変数と同様、その配列変数名の最後(カッコの直前)につけられた型宣言文字によって決まります。ただし、型宣言命令で一括指定を行った場合は、この限りではありません。詳しくは、「第2章 6.2 変数の型」を参照してください。

プログラム内で配列を使うには、あらかじめ DIM による“配列の宣言”が必要です。配列の宣言とは、変数の名前と配列に入れる要素の個数を決めてやることです。

DIM では次のように、その配列で使用可能な添字の最大値を指定することにより、要素の個数を決めます。

DIM A(5) 添字の最大値が5で、名前が“A”という配列変数を宣言

添字は、原則として0から始まりますので、実際の要素数は添字の値+1となります。この例では、5+1で、要素数は6個となります。

“添字が0から始まる”ということ、を、“添字の最小値 (OPTION BASE) が0である”といい、BASICの起動時にはこの状態になっています。なお、OPTION BASE という命令を使うことにより、添字の最小値を1にすることもできます。

また、DIM 中で配列変数を宣言する際、複数の添字をコンマで区切って指定すると、その配列変数は、添字の個数に応じた“次元”数をもつことになります。

たとえば、添字の最小値が0であるとして次のような配列を宣言した場合の、各配列の次元数および要素の合計数を示してみます。

DIM A(10) 1次元配列、要素数は11

DIM TA(2, 3) 2次元配列、要素数は $3 \times 4 = 12$

DIM NAME\$ (2, 5, 3) 3次元配列、要素数は $3 \times 6 \times 4 = 72$

2番目の例の場合、次のような2次元(縦方向と横方向の2次元)の配列がとられます。

TA(0, 0)	TA(0, 1)	TA(0, 2)	TA(0, 3)
TA(1, 0)	TA(1, 1)	TA(1, 2)	TA(1, 3)
TA(2, 0)	TA(2, 1)	TA(2, 2)	TA(2, 3)

配列変数の次元および添字の最大値は、メモリ容量が許す限り、いくつでも設定することができます。ただし、次元については、DIM で1行(255文字)中に記述できる範囲内、という制限もありますので注意してください。

なお、添字の最大値が10以下(添字の最小値が0でも1でもかまわない)のときは、DIM による宣言を行わなくとも、配列変数を用いることができます。

8. 予約語

N₈₈-BASIC(86)は、命令(関数も含む)の名前、演算子の名前などを、処理上、特殊なものとして扱います。これは“予約語”と呼ばれ、文字どおり BASIC によってその使用方法を予約されているものです。したがって、予約語をそのまま変数名として使用したりすることはできません。

予約語を使用する場合には、BASIC がその語を予約語であるか否か認識できるようにするため、語の前、後ろ、またはその両方に、BASIC 文法上で許された、あるいは規定された特殊文字(スペース、ダブルクォーテーション (")、ナンバー記号 (#)、コロン (:))などを挿入しなくてはなりません。

BASIC の予約語は、「付録 E. 予約語一覧」に掲げてあります。

9. 型変換

数値データは、必要に応じて他の型に変換することができます。型の変換は次の規則にしたがって行われます。

なお、文字型と数値型の間では型変換を行うことはできませんが、数値表記の文字列に限り、数値との相互変換を関数によって行うことができます(第3章中の STR\$, VAL の項参照)。

- (1) ある型の数値データが、違った型の数値変数に代入された場合、数値は、その変数名によって宣言された型に変換されます。

```
例)  10  ABC % = 1.234
      20  PRINT ABC %
      RUN
      1
```

- (2) 精度の違う数値間の演算の場合、精度の高い方に変換されて、演算が行われます。たとえば、10#/3の場合、10#/3#として演算が行われます。

```
例)  10  A # = 10# / 3
      20  B # = 10# / 3#
      30  PRINT A #, B #
      RUN
      3.333333333333333 3.333333333333333
```

- (3) 論理演算の場合、扱われる数値はすべて整数に変換され、結果は整数で与えられます。

```
例) 10 A=12.34
      20 B=NOT A
      30 PRINT B, A
      RUN
      -13          12.34
```

- (4) 実数が整数に変換される場合は、小数点以下は四捨五入されます。このとき、整数型で扱える範囲を超えた場合はエラーが起こります。

<pre>例) 10 A%=34.4 20 B%=34.5 30 PRINT A%, B% RUN 34 35</pre>	<pre>10 A#=1.234E+07 20 B%=A# 30 PRINT B%, A# RUN Overflow in 20</pre>
--	--

- (5) 倍精度実数型変数が単精度実数型変数に代入されたときは、変数の値は有効数字7桁に丸めたものとなります。単精度実数型変数の精度は7桁であり、もとの倍精度の数値との誤差の絶対値は、 $5.96E-8$ 以下となります。

```
例) 10 A#=1.23456789#
      20 B!=A#
      30 PRINT A#, B!
      RUN
      1.23456789  1.23457
```

倍精度実数型変数(あるいは倍精度実数型定数)と単精度実数型変数(あるいは単精度実数型定数)を混合して演算したり、単精度の値を倍精度実数型変数に代入したりすると、単精度値に変換する際、有効桁以降の桁に変換誤差が混入しますので注意してください。

例1) 精度の異なる数値による演算

- 好ましくない例 (演算結果に変換誤差が混入する)

```
A#=1.41421356#+0.12
```

- 改良例

```
A#=1.41421356#+0.12#
```

例2) 精度の高い変数に対する精度の低い値の代入

●好ましくない例

```
A#=3.1415
```

●改良例

```
A#=3.1415#
```

10. 式と演算

式とは、定数や変数を演算子（計算に使う特殊記号のこと）で結合した一般的な数式をはじめ、たんなる文字列や数値、変数だけのもの、または関数の総称です。たとえば次のものは、すべて式です。

```
例) 10+3/5
      A+B/C-D
      "BASIC"
      3.14
      A$
      TAN(D)
```

演算は、演算子または関数を用いて行う式の操作のことで、次の5つに分類されます。

- 1. 算術演算
- 2. 関係演算
- 3. 論理演算
- 4. 関数
- 5. 文字列演算

以下に、それぞれの演算について説明します。

10.1 算術演算

算術演算子には次のようなものがあり、示された順序にもとづいて演算を行います。なお、算術式の中に文字定数や文字変数が入ってはいけません。

	算術演算子	演算内容	例
実行 順序 ↓	^	指数(べき乗)演算	X^Y
	-	負号	-X
	*, /	乗算, 実数の除算	X*Y, X/Y
	+, -	加算, 減算	X+Y, X-Y

演算の実行順序を変更する場合は、カッコを使用します。カッコの中の演算子は他の演算より先に実行されます。カッコ内においては通常の実行順序に従います。

代数表記	BASIC の表記
1) $2X + Y$	$2 * X + Y$
2) $\frac{X}{Y} + 2$	$X / Y + 2$
3) $\frac{(X + Y)}{2}$	$(X + Y) / 2$
4) $X^2 + 2X + 1$	$X^2 + 2 * X + 1$
5) X^{Y^2}	$X^{(Y^2)}$
6) $(X^Y)^2$	$(X^Y)^2$
7) $Y(-X)$	$Y * -X$

なお、演算の実行順序についての詳細は、「第2章 10.6 演算の優先順位」を参照してください。

(1) 整数の除算と剰余の計算

整数の除算は \div によって行われます。扱われる数値が実数の場合は、演算が実行される前に小数点以下が四捨五入されます。商は小数点以下が切り捨てられた整数となります。

例) $10 \div 3 \rightarrow 3$ $(10 \div 3 = 3 \dots 1)$
 $23.75 \div 5 \rightarrow 4$ $(24 \div 5 = 4 \dots 4)$

剰余の演算は MOD によって行われます。扱われる数値が実数の場合は、演算が実行される前に小数点以下が四捨五入されます。結果は整数の割り算の余りです。

例) $13.3 \text{ MOD } 4 \rightarrow 1$ $(13 \div 4 = 3 \dots 1)$
 $25.68 \text{ MOD } 6.99 \rightarrow 5$ $(26 \div 7 = 3 \dots 5)$

(2) 0 での除算

式の実行時に 0 での除算が行われた場合は、“/0” (Division by zero) エラーを出力しますが、数値の型に応じて BASIC が扱うことのできる最大の数を結果として処理を続行します。

また、0 に対して負のべき乗演算を行った場合も同様になります。

例) PRINT 2# / 0	PRINT 0^-1
/0	/0
1.701411834604692D+38	1.70141E+38

(3) 桁あふれ (オーバーフロー)

代入や演算の結果がその変数の型内で表現することのできる範囲を超えた場合、桁あふれが発生します。

桁あふれが起こった場合、“OV” (Overflow) エラーを出力し、最大の数を結果として処理を続行します。

例) PRINT 3³⁰⁰

OV

1.70141E+38

10.2 関係演算

関係演算子は2つの数値を比較するときに用います。結果は、真(-1)、偽(0)で得られ、条件判定文などプログラムの流れを変えるのに用いられます (第3章中の IF…THEN~ELSE の項参照)。

関係演算子	演算内容	例
=	等しい	X=Y
<>, ><	等しくない	X<>Y, X><Y
<	小さい	X<Y
>	大きい	X>Y
<=, =<	小さいか等しい	X<=Y, X=<Y
>=, =>	大きい等しい	X>=Y, X=>Y

注意：=は代入文にも使うので注意すること。

IF…THEN~ELSE の中での使い方の例を次に示します。

例) IF X=0 THEN 1000

IF A+B<>0 THEN X=X+1:Y=Y+1

10.3 論理演算

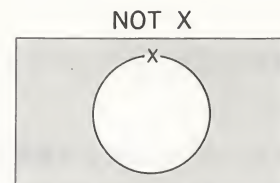
論理演算子は、ビット操作やブール演算(2進演算)を行ったり、複数の関係演算式を結合して複合条件を判定したりするのに用いられます。

論理演算子は、扱う数値を-32768から+32767までの2の補数表示の整数に変換してから、演算を行います。変換時にこの範囲外となれば“OV” (Overflow) エラーとなります。

各論理演算の内容は次のとおりです。論理演算は、扱う整数値の2進表記(16ビット分)に対してビットごとに行われます。次に示すのは、単一のビット X, Y についての演算結果を表しています。

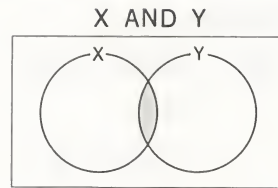
NOT : not (否定)

X	NOT X
1	0
0	1



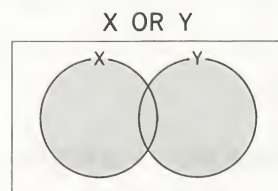
AND : and (論理積)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0



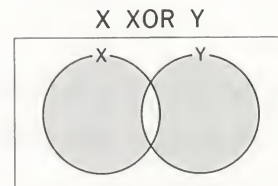
OR : or (論理和)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0



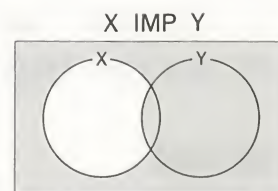
XOR : exclusive or (排他的論理和)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0



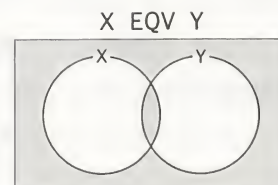
IMP : implication (包含)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1



EQV=equivalence (同値)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1



論理演算を利用することにより、バイトデータのあるビットパターンに照らし合わせて調べることができます。

たとえば、AND 演算子は機器の入出力ポートのステータスバイトの必要なビット以外のす

すべてのビットをマスクするのに使えます。また OR 演算子はある2進数を作るために2つのビットパターンを混合するのに使うことができます。

以下に、論理演算子がどのように働くかの例を示します。演算がビットごとに行われているようすに注意してください。

-1 OR 0 → -1	-1 = (1111 1111 1111 1111) ₂
	0 = (0000 0000 0000 0000) ₂
-1 OR 0 = (1111 1111 1111 1111) ₂	
48 AND 24 → 16	48 = (0000 0000 0011 0000) ₂
	24 = (0000 0000 0001 1000) ₂
48 AND 24 = (0000 0000 0001 0000) ₂	
15 XOR 60 → 51	15 = (0000 0000 0000 1111) ₂
	60 = (0000 0000 0011 1100) ₂
15 XOR 60 = (0000 0000 0011 0011) ₂	
17 EQV 12 → -30	17 = (0000 0000 0001 0001) ₂
	12 = (0000 0000 0000 1100) ₂
17 EQV 12 = (1111 1111 1110 0010) ₂	
28 IMP 9 → -21	28 = (0000 0000 0001 1100) ₂
	9 = (0000 0000 0000 1001) ₂
28 IMP 9 = (1111 1111 1110 1011) ₂	
NOT 23 → -24	23 = (0000 0000 0001 0111) ₂
NOT 23 = (1111 1111 1110 1000) ₂	

論理演算において、もし、0（偽）と-1（真）しか与えられなかったなら、扱う整数値の16ビット全部について同一の演算が行われることになり、演算の結果は0か-1のどちらかとなります（前例の-1 OR 0を参照）。したがって、論理演算子で2つ以上の関係演算を結ぶようにして使用することにより、複合条件によってプログラムの流れを変えるのに用いることができます。

例) IF X<0 OR 99<X THEN 1000

X が負または、99より大きければ、行番号1000へ飛ぶ。

IF 0<X AND X<100 THEN X=0

X が正でかつ、100より小さければ、X に0を代入する。

IF NOT (A=0) THEN 20

A が0でなければ、行番号20へ飛ぶ。

10.4 関数

関数とは、指定されたある値(これを関数の引数(ひきすう)という)に対して、ある決まった演算を行い、その演算の結果を得ることができるものです(ただし、関数によっては、引数を必要としないものもあります)。

関数は、演算子を使用する他の演算とは異なり、実行後にそれ自身が値をもつ、一種の命令として扱われます。したがって、ふつうの命令のように機能別に決められた名前をもっています。

N₈₈-BASIC (86) には、SIN (正弦), SQR (平方根) などの数値関数や CHR\$, LEFT\$ などの文字列関数といった“組み込み関数”が用意されています。これらの関数については、第3章で通常の命令とともに詳しく説明されています。

また、“ユーザー定義関数”としてユーザーが自由に定義できる関数機能もあります。これは第3章 DEF FN の項で説明します。

なお、初等関数 (SIN 関数などの数学関数) では、引数が整数や単精度のときは単精度の結果が得られますが、引数が倍精度のときは倍精度の結果を得ること (DISK モードのみ) ができます。

例) A=SIN(3.1416)+COS(3.1416)
PRINT 2, 2*2, SQR(2)

10.5 文字列の演算

BASIC では、文字列に対して、演算を行うことができます。

■文字列の連結

文字列は、演算子 “+” によって連結することができます。

例) 10 A\$="SUPER":B\$="PERSONAL":C\$="COMPUTER"
20 D\$=A\$+"-"+B\$+"-"+C\$
30 PRINT D\$
RUN
SUPER-PERSONAL-COMPUTER

■文字列の比較

文字も、数値の比較に用いられるものとまったく同じ関係演算子を用いて比較することができます。

=, <, >, <>, ><, <=, =<, >=, ==>

文字列の場合それぞれの文字列の最初から1文字ずつ、文字の比較を行います。もし相互にまったく同じ文字列の場合は、その2つの文字列は等しくなりますが、1個所でも違った場合は、その文字のキャラクタコード (「付録 C. キャラクタコード表」参照) の大きい方の文字列

が大きくなります。文字列の片方が短くて比較が途中で終わった場合は、短い文字列の方が小さくなります。

文字列の比較においては、空白なども意味をもちますから注意してください。

次の例は、すべて真（-1）となります。

例) "AA" < "AB"

"BASIC" = "BASIC"

"X&" > "X #"

"PEN " > "PEN"

"cm" > "CM"

"DESK" < "DESKS"

文字列の比較は、文字列の内容を調べたり、文字をアルファベット順に並べたり（ソート）するのに使うことができます。

10.6 演算の優先順位

種々の演算には優先順位があり、実行時において次の番号順に処理されます。

- | | |
|--------------------|----------------------|
| 1 カッコで囲まれた式 | 9 関係演算子 (<, >, = など) |
| 2 関数 | 10 NOT |
| 3 ^ (指数) | 11 AND |
| 4 - (負号) | 12 OR |
| 5 *, / (乗算, 実数の除算) | 13 XOR |
| 6 ÷ (整数の除算) | 14 IMP |
| 7 MOD (整数の剰余) | 15 EQV |
| 8 +, - (加算, 減算) | |

11. ファイル

11.1 ファイルとは

ファイルとは、意味を持つ情報の集まりです。ディスクに保存したプログラムやデータは、ファイルとして扱われます。また他の周辺機器との入出力も、まとまったデータの入出力という意味で、ファイルというと考えかたで扱うことができます。

N₈₈-BASIC(86)では、この考えかたにもとづき、ディスク装置をはじめとして、その他の周辺機器をも一括してファイルとしてみなすことにより、統一された入出力操作が可能となっています。

ファイルは“ファイルディスクリプタ”という一定の規則にしたがってつけられた名前によって区別されます。

11.2 ファイルディスクリプタ

ファイルディスクリプタは、周辺機器をも含むファイルを区別する名前で、次のような構成の文字列で表されます。

”[<デバイス名>:]<ファイル名>”

ファイルディスクリプタは文字列ですから、必ず、文字の並びをダブルクォーテーション(”)で囲んだ文字定数、または文字列を代入した文字変数の、いずれかでなければなりません。

■デバイス名

<デバイス名>はディスク装置や他の周辺機器（まとめて入出力機器という）の名称を表すもので、アルファベット 4 文字、もしくはアルファベット 3 文字と数字 1 文字に,”:”（コロン）をつけたものを原則としています。ただし、ディスク装置については、ドライブ番号を示す数字に”:”をつけた形が用いられます。

N₈₈-BASIC(86)で定義されているデバイス名は、次のとおりです。

デバイス名	入出力機器名	入力	出力
KYBD:	キーボード	○	×
SCRN:	スクリーン	×	○
LPT1:	プリンタ	×	○
CAS1:	カセットテープ (1200ボア)	○	○
CAS2:	〃 (600ボア)	○	○
1:	ディスクドライブ 1	○	○
2:	ディスクドライブ 2	○	○
}	}	}	}
14:	ディスクドライブ14	○	○
COM1:	RS-232C 第1回線	○	○
COM2:	RS-232C 第2回線	○	○
COM3:	RS-232C 第3回線	○	○

注意:

- KYBD:および SCRN:は、DISK モード BASIC でのみ指定可能です。
- CAS1:および CAS2:は、CMT インタフェースボードを実装している場合のみ指定可能です。
- 1:～14:は、ディスク装置の実装数分だけ指定可能です。
- COM2:, COM3:は、専用の RS-232C インタフェースボードを実装している場合のみ指定可能です。

〈デバイス名〉：は次のいずれかを指定する場合に限り、省略することができます。

- 1) ROM モード BASIC の場合のカセットテープ1 (CAS1：)
- 2) DISK モード BASIC の場合のフロッピーディスク1 (1：)

なお、RS-232C 回線 (COMn：) をファイルとして指定する場合は、OPEN により、ワード長やパリティチェックの有無などの、種々のモードを設定しなければなりません。詳しくは第3章中の OPEN の項を参照してください。

デバイス名についての詳細は、ユーザズマニュアルの「第10章 入出力装置とファイル」を参照してください。

■ファイル名

〈ファイル名〉とは、プログラムファイルやデータファイルにつける名前です。これはユーザーが指定します。ファイル名を必要とするのは、カセット、フロッピーディスクなどの補助記憶装置との入出力を行う場合のみで、その他の場合は省略することができます。ファイル名は次のような書式をとります。

〈ファイル名〉[.] [〈拡張ファイル名〉]

最初の〈ファイル名〉は6文字、ピリオドに続く〈拡張ファイル名〉は、もしあれば3文字までの文字列より構成されます。それぞれの文字数がそれ以下の場合には空いた部分に空白がうめられます。一般的には、〈ファイル名〉がファイルの名前を、〈拡張ファイル名〉がファイルの性質を表すようにしますが、とくにこれにこだわることなく自由に名前をつけてかまいません。

ふつうファイル名と言う場合は、〈拡張ファイル名〉まで含んだものを意味します。

ファイル名には1バイト系の英数カナ文字を使用できますが、コロン(:)およびキャラクタコード0と255で表される文字は使えません。

最初の〈ファイル名〉が6文字を超えて指定された場合、先頭から6文字目までの6文字が〈ファイル名〉として、続く9文字目までの3文字が〈拡張ファイル名〉としてみなされます。10文字目以降の文字は無視されます。

なお、カセットでは拡張ファイル名が使用できませんので、ファイル名は最長6文字までとなります。

11.3 データファイルとファイル番号

プログラム中で、データファイルを扱うときには、まず、OPEN によってファイルの使用を宣言します。

OPEN では、扱うファイルを〈ファイルディスクリプタ〉で指定して、〈ファイル番号〉を割り当てます。ファイル番号は入出力のために必要なメモリ領域(バッファ)に対してつける固有の

番号であり、以後、OPENされたファイルへの入出力はこのファイル番号を指示することによって行います。

N₈₈-BASIC (86) が起動したときに最初に指定するファイル数(「第1章 3. BASICの動作モード」参照)は、同時にOPENすることのできるファイルの数を意味しますが、このときに指定した数の範囲内でファイル番号を指定することができます。

データファイルの種類やその使い方などに関しては、第3章のファイル関連命令の各項、およびBASIC ユーザーズマニュアルを参照してください。

12. 割り込み

コンピュータの頭脳であるCPU (Central Processing Unit) は、一般に同時に2つ以上の処理をすることができません。したがって、1つ仕事を処理している間は他で何が起ころうと、CPUはその処理に移ることはできませんし、それを知ることさえできないのです。このことは、1つの決まった一連の流れを頭から処理するようなプログラムの場合、さほど問題にはなりませんが、1つのプログラムの処理中に、即実行しなければならないような特別なことがら(“事象”と呼ぶ)が起きたときに、その処理を先に実行させるようなプログラムを必要とする場合があります。

“割り込み”とは、このような処理を実現するための手段です。たとえば、CPUがある処理を実行しているとき、ファンクションキーが押されたなどの特別な事象が発生したとします。このような場合、専用のハードウェアはCPUにその事象が起きたことを、ハードウェア的手段で伝えます。そこでCPUは、現在実行中の処理を一時停止し、特別な事象の処理を優先的に実行し、その処理の終了後、元の処理に復帰するのです。

以上のような流れを、“割り込み”と呼ぶわけですが、PC-9801では、多くの周辺装置でこの割り込みを使える(すなわち、割り込みの事象を発生させることができる)ようになっており、N₈₈-BASIC(86)もこれらの割り込みをサポートしています。

次に、N₈₈-BASIC がサポートしている割り込み機能を示します。

- | | |
|----------------|----------------------|
| (1) STOP キー | (ON STOP GOSUB 参照) |
| (2) HELP キー | (ON HELP GOSUB 参照) |
| (3) リアルタイム・タイマ | (ON TIME\$ GOSUB 参照) |
| (4) ファンクションキー | (ON KEY GOSUB 参照) |
| (5) RS-232C 回線 | (ON COM GOSUB 参照) |
| (6) ライトペン | (ON PEN GOSUB 参照) |

13. ラベル名

N₈₈-BASIC(86)では、プログラムの分岐先やプログラムの編集時などに利用する行番号の代わりとして、“ラベル名”を用いることができます。

あるルーチンの始まりなどに意味のあるラベル名をつけておけば、いちいち行番号を覚えておかなくともよく、また RENUM で行番号のつけ替えを行ってもそのたびに新しい行番号を確かめる必要もなくなるため、プログラムの作成を非常に楽に行うことができます。後でプログラムの修正などを行うときにもたいへん有利です。

まず、ラベル名を使わないで、ふつうに行番号を分岐先として用いたプログラムを例として示します。

```
10 INPUT A
20 IF A<0 THEN 80
30 IF A>0 THEN 60
40 PRINT "zero"
50 GOTO 90
60 PRINT "plus"
70 GOTO 90
80 PRINT "minus"
90 END
```

これは、10行で入力された値が正か負か0かを調べるプログラムです。ここで処理の流れを変える命令が20, 30, 50, 70行で使われていて、その分岐先の指定はすべて行番号になっています。

これを、ラベル名を使って書き換えると次のようになります。

```
10 INPUT A
20 IF A<0 THEN *MINUS
30 IF A>0 THEN *PLUS
40 PRINT "zero"
50 GOTO *EXIT
60 *PLUS : PRINT "plus"
70 GOTO *EXIT
80 *MINUS : PRINT "minus"
90 *EXIT : END
```


プログラムが見やすくなり、処理の内容もよくわかるようになりました。

このように、ラベル名とは、分岐先の目印として自由につけることのできるものです。

ラベル名の使用については、次の注意を守らなければなりません。

- (1) ラベル名の頭には必ずアスタリスク(*)をつけなければなりません。
- (2) 先頭のアスタリスクを除いて、ラベル名は必ず1バイト系の英文字で始まらなければなりません。
- (3) ラベル名に使用できる文字は、先頭のアスタリスクを除いて1バイト系英文字と数字とピリオド(.)であり、大文字と小文字の区別はありません。
- (4) 予約語をラベルとして使用することはできません。ただし、中に含む場合はかまいません。
- (5) ラベル名の長さは、プログラムの1行の範囲(255バイト以内)によってのみ制限を受けます。
- (6) 参照されるラベル名(呼ばれる方のラベル名)は、必ず行の最初になければなりません。
- (7) 1行中でラベル名の後に続けて命令を記述し、マルチステートメントとするときは、コロン(:)またはスペースによって区切ります。

以上のような制限を無視すると“Syntax error”となります。

その他参照されるラベル名に同じものがあつた場合、二重定義されているという意味の“Duplicate label”エラーが生じます。これらのエラーの検出は、“RUN”したとき、プログラムの実行に先だつて行われますから、ラベル名のエラーがあるとプログラムを1行も実行することができません。

また、この場合のエラーメッセージはエラー箇所を示す〈行番号〉はともないません。

ラベル名は、プログラム中で分岐の目印として使用できるほか、LIST、DELETEなど〈行番号〉を対象とする命令のパラメータにはすべて使用することが可能です。

例) LIST *START—*LAST
DELETE *LOOP1—*EXIT1

14. エラーメッセージ

N₈₈-BASIC(86)は、プログラムの実行を中断させなければならないようなエラーを実行時に検出したとき、エラーメッセージを画面に出力し、コマンドレベルにもどります。

ダイレクトモードの場合、エラーメッセージは次のような形式で出力されます。

XXXX...

プログラムモードの場合は次のように出力されます。

```
XXXX...in yyyy
```

XXXX...はエラーメッセージで、yyyy はエラーが検出された行番号です。この行はあくまでも、BASIC がエラーを検出した行であり、真のエラーの原因は他の行にある場合がありますのでご注意ください。

エラーメッセージの内容とその対策については、「付録A. エラーメッセージとその対策」を参照してください。

第3章

3

命令リファレンス

第3章 命令リファレンス

この章の見方

この章では、すべての命令（関数も含む）について書式、機能などを解説しています。各命令の解説は次の構成で行われています。

1 → **KACNV\$** (DISK モード) ← 2 3 → 関 数

4 → **機 能** 2 バイト系全角文字を、対応する 1 バイト系の英数カナ文字に変換します。

5 → **書 式** KACNV\$(〈文字列〉)

6 → **文 例** A\$=KACNV\$(B\$)
PRINT KACNV\$("アイウ A B C")

7 → 〈文字列〉中の 2 バイト系全角文字を 1 バイト系英数カナ文字に変換します。なお、このとき 2 バイト系全角文字列中の KI/KO コードはすべて取り除かれます。
なお、〈文字列〉中に、対応する 1 バイト系英数カナ文字のない 2 バイト系全角文字が含まれていると、“Illegal function call” エラーとなります。

8 → **注意：**〈文字列〉中の 2 バイト系半角文字は正しく変換されません。

9 → **参照：**AKCNV\$, サンプルプログラム 35

1. 命令の名前。
 2. DISK モード BASIC でのみ有効な命令であることを示します。
 3. 関数表示。該当命令が関数である場合のみ、“関数” と明示してあります。
 4. 機能。命令の機能を簡単に示します。
 5. 書式。命令の記述の仕方を示します。実際の入力時には次のような決まりに従ってください。
- アルファベットの太文字で示された項目は、そのまま 1 バイト系のアルファベットを入力します。入力する場合は、小文字でも大文字でもかまいません。BASIC は入力された命令（行）を、自動的にすべて大文字に変換します。ただし、ダブルクォーテーション (") で囲まれた文字列や、DATA 行中に記述された文字列などは、大文字と小文字の区別がなされますから、必要に応じて使い分けてください。
 - カギカッコ (〈 〉) で囲まれた項目は、ユーザーが指定します。
 - 角カッコ ([]) で囲まれた項目は、オプションであり省略することができます。省略した場合、デフォルト値 (BASIC によって設定される値) または以前に指定した値が適用されます。

コンマ（,）で区切られる複数のパラメータがあり、しかも省略可能なパラメータがある場合、次の例のような書式で示します。

```
CLEAR  [<ダミーパラメータ>] [, <メモリの上限>] [, <スタックの大きさ>] [, <配  
列データ領域の大きさ>]
```

この場合、角カッコで囲まれたパラメータを省略することが可能であることを示しています。ここで、あるパラメータ以後をすべて省略する場合は、コンマも含めて省略できます。しかし、中途のパラメータを省略したうえで後続のパラメータを指定する場合は、それ以前のコンマはすべて指定する必要があります。

たとえば、上記の例で〈スタックの大きさ〉のみ指定する場合は、次のようにします。

```
CLEAR  ,, <スタックの大きさ>
```

なお、次のようにパラメータの後にコンマまたはセミコロン（;）を指定するような場合は、コンマならびにセミコロンも含めて省略します。

```
INPUT  [<プロンプト文> | ; | ] <変数> [, <変数> ...]  
      , |
```

ただし、この約束と異なっている場合もあります。その場合には、注意書きにてそのことを説明します。

- 上記のカギカッコ、角カッコ以外の記号で丸カッコ（），コンマ（,），セミコロン（;），マイナス記号（-），等号（=）などの記号は示された位置に正しく入力します。
- 省略記号“…”の続く項目は、1行の許す長さ（255文字）の内で任意の回数繰り返すことができます。たとえば、

```
DATA  <定数> [, <定数> ...]
```

という書式ならば、次のような記述が可能です。

```
DATA  0, 10, 15, 20
```

- 座標指定の所で（Wx, Wy）と記されているのはワールド座標を、（Sx, Sy）はスクリーン座標を表しています。またSTEP（x, y）という記法は相対座標による指定ができることを意味しています。その他たんに（X, Y）と記されているのはキャラクタ座標を表しています。
- 書式の中で〈行番号〉またはそれに類する行番号の指定をさしているものは、行番号の他にラベル名も含んでいると解釈してください。

6. 文例. 実際の入力の方の見本として簡単な例を示します。
7. 解説. 命令の使用法や詳しい機能を説明します。
8. 注意. 命令を使ううえで、特に注意すべきことをまとめています。
9. 参照. 関連の深い他の命令と、第4章中のサンプルプログラムの番号を示します。

■ DISK モードについて

N₈₈-日本語 BASIC(86) システムディスクから BASIC を起動した場合を“DISK モード”と呼び、ROM に搭載されている N₈₈-BASIC(86) を起動した場合を“ROM モード”と呼びます(「第1章 1. ROM モード BASIC と DISK モード BASIC」参照)。

大部分の命令は両 BASIC で使用可能ですが、ディスクファイルを扱う命令をはじめとする一部の命令は DISK モード BASIC でのみ使用することができます。この章では、この DISK モードでのみ使える命令には、命令の名前の後に、

(DISK モード)

と明記し、区別できるようにしてあります。

なお、算術関数の中で、初等関数(SIN, COS, TAN などの数学関数)は引数として倍精度実数型の数値を与えると倍精度実数型の値を、その他の数値を与えると単精度実数型の値を得ることができますが、倍精度関数機能は DISK モード BASIC でしか使うことができませんので注意してください。

ABS

関 数

機 能 絶対値を得ます。

書 式 ABS(<数式>)

文 例 B=ABS(-2)
PRINT ABS(-1.0000000000000001)

<数式> に指定した値の絶対値を得ます。
<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：サンプルプログラム 8

AKCNV\$ (DISK モード)

関 数

機 能 1 バイト系の英数カナ文字を、対応する 2 バイト系全角文字に変換します。

書 式 AKCNV\$(<文字列>)

文 例 A\$=AKCNV\$(B\$)
PRINT AKCNV\$("イロハ日本語 ABC 英語")

<文字列> 中の 1 バイト系文字を 2 バイト系全角文字に変換します。<文字列> 中の 2 バイト系日本語文字に対しては変換操作を行いません。

参照：KACNV\$, サンプルプログラム 35

ASC

関 数

機 能 文字のキャラクタコードを得ます。

書 式 ASC(<文字列>)

文 例 A=ASC("NEW")
PRINT ASC(A\$)

<文字列> 中の最初の文字のキャラクタコードを、10 進表記で得ます。

参照：CHR\$, サンプルプログラム 23

ATN

関 数

機 能 逆正接(アークタンジェント)を得ます。

書 式 ATN(<数式>)

文 例 ANGLE=ATN(Y/X)

PRINT ATN(-3.14159/2)

<数式>の値に対する逆正接値を得ます。得られる値はラジアンで、 $-\pi/2$ から $\pi/2$ までの範囲です。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照: TAN

ATTR\$(DISK モード)

関 数

機 能 ファイル、ドライブの属性を得ます。

書 式 ATTR\$(
 <ドライブ番号>
 #<ファイル番号>
 <ファイルディスクリプタ>
)

文 例 PRINT ATTR\$(1)

A\$=ATTR\$("1: TOOL.BAS")

指定したドライブ、オープンされているファイル、ディスク上のファイルの現在の属性文字を得ます。属性文字は以下の意味を持ちます。

"□□□": 属性は解除されており、通常を読み出し、書き込みが可能な状態です。

"R□□": 書き込みの際に、リードアフターライト(書き込んだ内容とメモリ上の内容の比較チェック)を行います。

"□□P": 書き込みが禁止されています。

"□E□": P オプションつきでSAVEされたファイルです。

参照: SAVE, SET

AUTO

機 能	行番号を自動的に発生します。
書 式	AUTO [〈行番号〉] [〈増分〉]
文 例	AUTO 100, 5

AUTO を実行すると〈行番号〉で指定した行から、以後、リターンキーの入力ごとに〈増分〉ずつ増加した行番号を自動的に発生します。

〈行番号〉、〈増分〉を省略した場合には、ともに 10 を採用します。コンマがあつて〈増分〉を省略した場合には、直前に実行した AUTO コマンドの増分が用いられます。

AUTO を中止してコマンドレベルにもどるには **CTRL** + **C** または、**STOP** キーを押します。このとき、最後に発生した行番号の行は、メモリ中にプログラムとして格納されません。

プログラム中にすでに存在する行と同じ行番号を発生させた場合には、行番号の直後にアスタリスク(*)が表示され、注意を促します。このとき、文字を入力してリターンキーを押すと、その行は新しい内容に入れ代わります。また、何も入力しないでリターンキーを押した場合には、その行は削除されます。

BEEP

機 能	内蔵スピーカを鳴らしたり、止めたりします。
書 式	BEEP [〈スイッチ〉]
文 例	BEEP

〈スイッチ〉の値が 1 の場合は ON (鳴りっぱなし)、0 の場合は OFF (鳴りやむ) となります。〈スイッチ〉を省略した場合には、PRINT CHR\$(7) を実行したのと同様に、一定時間スピーカを鳴らします。

参照：サンプルプログラム 31

BLOAD (DISK モード)

機 能	機械語ファイルをメモリ上にロードします。
書 式	BLOAD <ファイルディスクリプタ> [〈ロードアドレス〉] [R]
文 例	BLOAD "SUB1", R BLOAD "SUB2", &H100

〈ファイルディスクリプタ〉には、メモリ上にロードしたい、ディスク上または RS-232C 回線上の機械語ファイル(バイナリイメージのファイル。プログラムでもデータでもかまわない)を指定します。

〈ロードアドレス〉を指定した場合には、直前に実行された DEF SEG で指定されたセグメントベースに、〈ロードアドレス〉で与えられたアドレス(相対アドレス)を加えた番地からロードし始めます。

〈ロードアドレス〉を省略した場合には、直前に実行された DEF SEG で指定されたセグメントベースに、機械語ファイルをセーブする際に BSAVE で指定した開始アドレス(相対アドレス)を加えた番地からロードが行われます。

R オプションを指定すると、機械語ファイルをロード後、直ちにその先頭番地からプログラムとして実行を開始します。したがってこの場合、指定する機械語ファイルは、実行可能なプログラムでなくてはなりません。

R オプションを指定し、さらに〈ロードアドレス〉をつけてロードする場合には、セーブされた際と異なる番地にロードされても実行可能な性質をもった(リロケータブルな)プログラムを、機械語ファイルとして指定しなければなりません。

なお、R オプションの指定時には、すでに開かれているデータファイルはその状態を保持します。

参照：BSAVE, CLEAR, DEF SEG, LOAD

BSAVE (DISK モード)

機 能 メモリ上の指定範囲の内容を、ディスク上あるいは RS-232C 回線上に機械語ファイルとしてセーブします。

書 式 BSAVE 〈ファイルディスクリプタ〉, 〈開始アドレス〉, 〈長さ〉

文 例 BSAVE "SUB1", &H100, &H2FF

〈ファイルディスクリプタ〉には、メモリ上の内容をセーブする際における、ディスク上あるいは RS-232C 回線上のファイル名を指定します。

〈開始アドレス〉には、セーブを開始する番地を相対アドレスで指定します。

〈長さ〉には、セーブするメモリ上の内容のバイト数を指定します。

BSAVE は、直前に実行された DEF SEG で指定されたセグメントベースに、〈開始アドレス〉の値を加えた番地以降の連続する〈長さ〉バイトの内容を、機械語ファイルとしてセーブします。

参照：BLOAD, CLEAR, DEF SEG, SAVE

CALL

機 能 メモリ上に用意された機械語サブルーチン呼び出し、実行します。

書 式 CALL <変数名> [(<引数> [, <引数> ...])]

文 例 CALL MUSBR(X, Y)
CALL M(ARG1\$, ARG2\$, RESULT\$)

呼び出したい機械語サブルーチンは、あらかじめメモリ上に用意しておかなくてはなりません。このためには、POKE や BLOAD を用いることができます。

<変数名>には、呼び出したい機械語サブルーチンの実行開始アドレス(相対アドレス)が代入された、変数の名前を指定します。ここで用いる変数には配列変数を用いることはできません。

<引数>には、機械語サブルーチンに渡す変数を指定します。引数としてはすべての型の変数を指定することができますが、定数や式の指定はできません。

CALL を実行すると、直前に実行された DEF SEG によって指定されたセグメントベースに、<変数名>で指定した変数に代入されている相対アドレス値を加えた番地に実行が移ります。CALL によって呼び出されたサブルーチンは、機械語の IRET 命令により BASIC に制御をもどすことができます。

<引数> の受け渡し方法については、BASIC ユーザーズマニュアルを参照してください。

参照：BLOAD, BSAVE, CLEAR, DEF SEG, POKE

CDBL

関 数

機 能 整数値、単精度実数値を、倍精度実数値に変換します。

書 式 CDBL(<数式>)

文 例 A#=CDBL(B!/2)
A#=CDBL(256%)

<数式>の値を倍精度実数値に変換した値を得ます。ただし、型変換が行われるだけで有効桁数の変化がありませんから、得られた値の精度は、変換する前の型と同じ(整数型なら整数部のみ、単精度実数型なら有効数字 6 桁)になります。

参照：CINT, CSNG

CHAIN (DISK モード)

機 能	メモリ上のプログラムからディスク上のプログラムに実行を移します。
書 式	CHAIN [MERGE] <ファイルディスクリプタ> [, <行番号>] [, ALL] [, DELETE <範囲>]
文 例	CHAIN MERGE "TEST.BAS", 1000, ALL, DELETE 500-600

CHAIN を実行すると、メモリ上にあるプログラムで使用中のファイルは、そのままの状態ロードされるプログラムに引き継がれます。また、OPTION BASE もロードされるプログラムに引き継がれます。

(1) MERGE オプションを省略した場合

メモリ上のプログラムは消去され、ディスクから <ファイルディスクリプタ> で指定したプログラムがロードされ、実行されます。

<行番号>には、ロードされたプログラムの開始行番号を指定します。省略した場合は、プログラムの最初から実行されます。

ALL オプションは、連結後のプログラムに変数、配列を引き渡すのに使います。これを指定した場合すべての変数、配列が引き渡され、省略した場合いっさい引き渡されません。必要な変数、配列だけを引き渡したいという場合は COMMON を使います (COMMON 参照)。

DELETE オプションは、この場合意味をもちません。

(2) MERGE オプションを指定した場合

メモリ上のプログラムと、ディスクからロードされたプログラムとが、連結(マージ)されてひとつのプログラムとなった後、実行されます。

<ファイルディスクリプタ>には、ディスクからロードするプログラムを指定します。この場合、ディスクからロードするプログラムは、必ずアスキーセーブされたものでなければなりません。

<行番号>には、連結後のプログラムの開始行番号を指定します。行番号を省略した場合には、連結後のプログラムの最初から実行されます。

ALL オプションの扱いは、(1)と同様です。

DELETE オプションおよび<範囲>を指定すると、メモリ上のプログラムの指定範囲が削除された後、ディスクのプログラムの連結が行われます。<範囲>は、"200-300" のように、行番号をマイナス記号でつないで指定します。DELETE オプションで指定する<範囲>の行番号は RENUM を実行すると書き換えられます。また、<範囲>の指定には、ラベル名を使うことができます。

注意：実行開始行を指定する〈行番号〉は、RENUM を実行しても書き換えられません。また、〈行番号〉としてラベル名を使うことはできません。

DEFINT, DEFSNG, DEFDBL, DEFSTR などの宣言文は、MERGE オプションを指定した場合、連結後のプログラムに対しても有効となりますが、指定がない場合、ロードされたプログラムには何の効果も与えません。

プログラムを連結すると、DEF FN, あるいは ON COM GOSUB, ON ERROR GOTO などの割り込み制御は無効となります。

なお、OPTION BASE は、連結後のプログラムに引き継がれます。

参照：COMMON, サンプルプログラム 1, 2

CHR\$

関 数

機 能 指定したキャラクタコードを持つ文字を得ます。

書 式 CHR\$(〈数式〉)

文 例 A\$=CHR\$(65)
PRINT CHR\$(&H4E)

〈数式〉の値のキャラクタコードを持つ文字を得ます。値が 0～255 の範囲にない場合は、“Illegal function call” エラーになります。

参照：ASC, サンプルプログラム 23

CINT

関 数

機 能 単精度実数値, 倍精度実数値を, 整数値に変換します。

書 式 CINT(〈数式〉)

文 例 A%=CINT(B! * 2)
A%=CINT(X #)

〈数式〉の値の小数点以下を四捨五入して整数に変換した値を得ます。結果の値が -32768～32767 の範囲にない場合は、“Overflow” エラーになります。

参照：CDBL, CSNG

CIRCLE

機能

円, 楕円を描きます。

書式

CIRCLE (Wx, Wy) , <半径> [, <パレット番号 1>] [, <開始角度>] [, <終了角度>] [, <比率>] [, F [, <パレット番号 2>]]
STEP(x, y) <タイルストリング>

文例

CIRCLE (80, 80), 40, 4, 0, 6.28

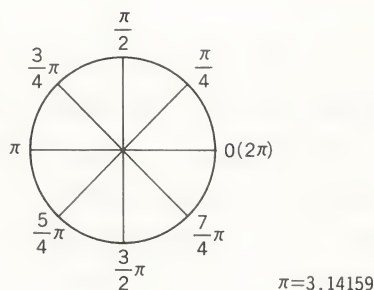
CIRCLE STEP(40, -20), 30,,,,, F, TILE\$

ワールド座標の(Wx, Wy)を中心とし、<半径>で指定される大きさの円を描きます。中心座標は、STEPをつけてLP(最終参照点)からの相対座標により指定することもできます。

<パレット番号 1>は、描く円の色をパレット番号で指定します。省略された場合には、COLORで指定されているフォアグラウンドカラーのパレット番号が用いられます。

<開始角度>、<終了角度>を指定すると、指定された角度の範囲内のみに円弧を描きます。角度には $-2\pi \sim 2\pi$ の範囲(π は円周率)のラジアン値を指定します。省略するとそれぞれ、0と 2π が採用されます。<開始角度>、<終了角度>が負であった場合には、その絶対値が用いられますが、そのときには、中心から半径線が描かれますので扇形を描くことができます。

<比率>は、円の偏平率を(垂直方向の半径)/
(水平方向の半径)で指定します。これを指定することで、楕円を描くことができます。ただし、 640×400 ドットのモードでは実際に表示される画面上の長さの比率で指定し、 640×200 ドットのモードではその $1/2$ の値を指定します。省略された場合には、 640×400 ドットのモードでは1.0、 640×200 ドットのモードでは0.5が用いられます。



<比率>が1以外(640×400 ドットのとき、 640×200 ドットのときは0.5以外)の場合、垂直方向の半径と水平方向の半径のうち大きい方に<半径>が用いられます。つまり、 640×400 ドットのモードの場合、<半径>が同じでも<比率>が1以下ならば横長の楕円となり、1以上ならば縦長の楕円となります。

Fを指定すると、円を描くのと同時にその内部を<パレット番号 2>で指定されたパレットの色または<タイルストリング>((2)PAINT 参照)により指定された模様でぬりつぶします。<パレット番号 2>および<タイルストリング>ともに省略された場合には、円を描いたパレットの色でぬりつぶします。開始角度、終了角度が指定されているときにFを指定した場合には、扇形が描かれ内部がぬりつぶされます。

CLEAR

注意：CIRCLE を実行すると、LP(最終参照点)は円の中心座標(Wx, Wy)に設定されます。

参照：COLOR, [2] PAINT, サンプルプログラム 14, 18, 21

CLEAR

機能	変数の初期化およびメモリレイアウトを決定します。
書式	CLEAR [<ダミーパラメータ>] [<メモリの上限>] [<スタックの大きさ>] [<配列データ領域の大きさ>]
文例	CLEAR , &H9E00, &H400

すべての数値変数を 0 に、また文字変数を""
(空の文字列、ヌルstringともいう)に初期
化します。また、DEF(DEF FN, DEFINTなど)
によって定義あるいは指定された情報もすべて
無効にします。

<ダミーパラメータ>は値を指定しても何の効
果もないものですから省略してかまいません
が、CLEAR 直後のコンマ(,)は必ず入れてくだ
さい。

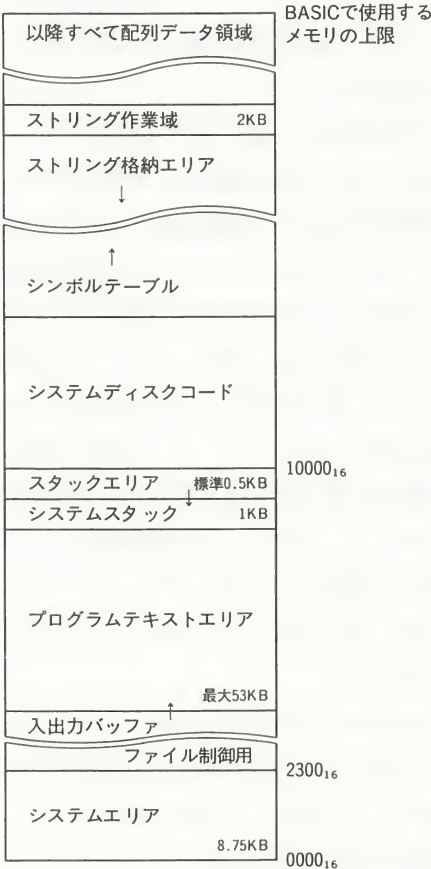
<メモリの上限>は、セグメントベース(実際
の物理番地を 16 で割った値)で指定します。指
定された番地の直前まで、すなわち(指定値 *
16-1)番地までを BASIC の使用するメモリの
上限値とします。その番地以降に置かれたデー
タや機械語プログラムは、BASIC によってク
リアあるいは破壊されることはありません。

<スタックの大きさ>は、BASIC が FOR,
GOSUB, PAINT など使用するスタック領
域の大きさをバイト数で指定します。リセット
時の初期化された状態では 512 バイトに設定さ
れています。

<配列データ領域の大きさ>には、数値配列の値が格納される配列データ領域のサイズを指定
します。

実際に確保される領域の大きさは、それぞれ指定した値の 16 倍となります。

参照：BLOAD, BSAVE, FRE



N88-日本語BASIC(86)のメモリレイアウト概要

CLOSE

機 能 ファイルを閉じます。

書 式 CLOSE [[#] <ファイル番号> [, [#] <ファイル番号>] ...]

文 例 CLOSE

CLOSE #1, #3

CLOSE は、OPEN によって開かれていたファイルを閉じます。閉じられたファイルに対しては、再び開かれるまで入出力を行うことはできません。

<ファイル番号>を指定すると、OPEN で指定したファイル番号に対応するファイルを閉じます。<ファイル番号>を複数個指定すれば、複数のファイルをいちどに閉じることができます。<ファイル番号>を省略した場合には、そのとき開いているファイルすべてを閉じます。

閉じられたファイルにつけられていたファイル番号は、同じあるいは異なるファイルを開くために再び使うことができます。また閉じられたファイルは、同じあるいは異なったファイル番号によって再び開くことができます。

ファイルが出力用に開かれていた場合には、CLOSE を実行すると、バッファに残っていたデータが完全書き出されます。このため、ファイルの出力処理を正しく終了するには、CLOSE の実行が必要です。

R 指定なしの RUN および R 指定なしの LOAD は、プログラムの実行を開始する前に、開かれているすべてのファイルを自動的に閉じます。また、END あるいは NEW の実行によってもファイルは自動的に閉じられます。

注意：STOP ではファイルを閉じることにはできません。また、END が実行されずにプログラムが終了する場合もファイルは閉じられません。

参照：CHAIN, END, LOAD, NEW, OPEN, STOP, サンプルプログラム 26, 27, 28, 29

CLS

機 能 現在アクティブな画面をクリアします。

書 式 CLS [<機能>]

文 例 CLS 2

<機能>は 1, 2, 3 の値を取り、それぞれ次のように働きます。省略された場合には 1 が採用されます。

- 1 : テキスト画面のみをクリアします。テキスト表示モードが CONSOLE, [1] COLOR により、白黒リバースモードになっている場合には画面は白くなります。
- 2 : グラフィック画面のビューポート内をクリアします。グラフィック表示が SCREEN によりカラーモードに設定されている場合には、[1]COLOR により指定されたバックグラウンドカラーで、ビューポート内をクリアします。
- 3 : テキスト画面、グラフィック画面の両方をクリアします。

テキスト画面をクリアする場合、CONSOLE によって指定されたスクロールウィンドウ内をクリアします。また、グラフィック画面をクリアした場合には、LP(最終参照点)はビューポートの左上の頂点に移動します。

参照：[1] COLOR, CONSOLE, SCREEN, VIEW

[1] COLOR

機 能 ディスプレイ画面の各部の色およびグラフィック画面のパレットモードを指定します。

書 式 COLOR [<ファンクションコード>] [, <バックグラウンドカラー>] [, <ボーダーカラー>] [, <フォアグラウンドカラー>] [, <パレットモード>]

文 例 COLOR 7, 0, 0, 7

[1]COLOR は、テキスト画面の文字のカラーを変えたり、グラフィック画面のフォアグラウンド、バックグラウンド、ボーダーの各カラーを設定したり、グラフィック画面のパレットモードを指定したりするのに使用します。

<ファンクションコード>は、テキスト画面の文字にいろいろな機能を与えます。このファンクションコードは、テキスト画面がカラーモードになっているか、白黒モードになっているかによって、働きが異なります。なお、カラーモード／白黒モードの切り替えは CONSOLE で行います。<ファンクションコード>で指定する値は、“パレット番号”ではありませんので、[2]COLOR によってカラーパレットの変更を行っても色は変化しません。

白黒モードの場合(CONSOLE ,,, 0)

- 0 : ノーマル(通常の表示)
- 1 : シークレット(文字は表示されない)
- 2 : ブリンク(点滅する)
- 3 : シークレット(1と同じ)
- 4 : リバース(反転する)

- 5 : リバースシーケレット (反転して文字は表示されない)
- 6 : リバースブリンク (反転して点滅する)
- 7 : リバースシーケレット (5 と同じ)

カラーモードの場合(CONSOLE ,,, 1)

- | | | | |
|-------|--------|-------|-------|
| 0 : 黒 | 1 : 青 | 2 : 赤 | 3 : 紫 |
| 4 : 緑 | 5 : 水色 | 6 : 黄 | 7 : 白 |

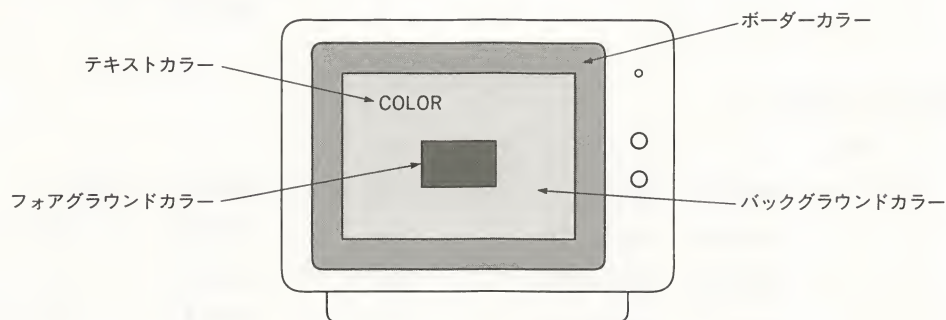
〈バックグラウンドカラー〉は、グラフィック画面の地の色を表します。このパラメータは、"パレット番号" (〈パレットモード〉の項および〔2〕COLOR 参照) によって指定します。このパラメータの設定後 CLS によって画面をクリアすると、設定色によって画面がぬり変えられます。また、以後 PRESET を色指定なしで実行すると、この色が採用されます。

なお、白黒モード (高分解能白黒モードを含む) では、〈バックグラウンドカラー〉として指定されたパレット番号は、0 のときは黒、0 以外のときは白とみなされます。

〈ボーダーカラー〉は、画面の中で BASIC によって使うことのできる領域外の、枠の色を表すもので、次のような値で指定します。

- | | | | |
|-------|--------|-------|-------|
| 0 : 黒 | 1 : 青 | 2 : 赤 | 3 : 紫 |
| 4 : 緑 | 5 : 水色 | 6 : 黄 | 7 : 白 |

〈ボーダーカラー〉が指定されると、カラーモードでも白黒モードでも、画面の外枠に色 (白黒のディスプレイでは濃淡) がついて表示されます。なお、このパラメータは、専用高解像度ディスプレイの使用時には意味がありません。



〈フォアグラウンドカラー〉は、グラフィック画面に点や線を表示したりするときに使われる色を表します。このパラメータは、"パレット番号" によって指定します。種々のグラフィック命令 (PSET, LINE, CIRCLE など) でとくに色指定をしなかった場合、この色が採用されます。

なお、白黒モード (高分解能白黒モードを含む) では、〈フォアグラウンドカラー〉として指定されたパレット番号は、0 のときは黒、0 以外のときは白とみなされます。

〈パレットモード〉は、グラフィック画面に対する色指定のモードを指定します。〈パレットモード〉に指定できる値とその意味は次のとおりです。

[1] COLOR

- 0 : 8 色中・8 色モード。8 個のパレットにシステムが決めた 8 色を対応づけるモードです。
- 1 : 4096 色中・8 色モード。8 個のパレットに 4096 色中の任意の 8 色を対応づけるモードです。
- 2 : 4096 色中・16 色モード。16 個のパレットに 4096 色中の任意の 16 色を対応づけるモードです。

N₈₈-BASIC(86)の起動直後のパレットモードは 8 色中・8 色モードです。以降、〈パレットモード〉が指定された場合に限りパレットモードが切り替わります。各モードにおけるパレットとカラーコードの対応づけは、[2] COLOR によって設定されますので、参照してください。

〈パレットモード〉が指定されてパレットモードが切り替わると、そのたびにパレットとカラーコードの関係は次のように初期化されます。

8 色中・8 色モード

〈パレット番号〉 〈カラーコード〉

0	0	(黒)
1	1	(明るい青)
2	2	(明るい赤)
3	3	(明るい紫)
4	4	(明るい緑)
5	5	(明るい水色)
6	6	(明るい黄)
7	7	(白)

4096 色中・8 色モード

〈パレット番号〉 〈カラーコード〉

0	&H000	(黒)
1	&H00F	(明るい青)
2	&H0F0	(明るい赤)
3	&H0FF	(明るい紫)
4	&HF00	(明るい緑)
5	&HF0F	(明るい水色)
6	&HFF0	(明るい黄)
7	&HFFF	(白)

4096 色中・16 色モード

〈パレット番号〉 〈カラーコード〉

0	&H000	(黒)	8	&H777	(灰色)
1	&H00F	(明るい青)	9	&H00A	(少し暗い青)
2	&H0F0	(明るい赤)	10	&H0A0	(少し暗い赤)
3	&H0FF	(明るい紫)	11	&H0AA	(少し暗い紫)
4	&HF00	(明るい緑)	12	&HA00	(少し暗い緑)
5	&HF0F	(明るい水色)	13	&HA0A	(少し暗い水色)
6	&HFF0	(明るい黄)	14	&HAA0	(少し暗い黄)
7	&HFFF	(白)	15	&HAAA	(少し暗い白)

注意：4096 色中・8 色モードおよび 4096 色中・16 色モードはアナログ RGB 対応ディスプレイが接続されている場合のみ有効です。アナログ RGB 対応ディスプレイが接続されてい

ない場合でも、エラーにはなりません、指定どおりの色は出ません。

参照：〔2〕 COLOR, COLOR@, CONSOLE, サンプルプログラム 15, 16

〔2〕 COLOR

機 能	カラーパレットの色を変更します。
書 式	COLOR[(〈パレット番号〉, 〈カラーコード〉)]
文 例	COLOR=(2, 4) COLOR=(PAL, COL)

グラフィック画面への色の指定は、すべてカラーパレットによって行いますが、〔2〕COLORは、このカラーパレットの色を決める命令です。

〈パレット番号〉には0から7(あるいは0から15)の8個(あるいは16個)のカラーパレットの固有の番号を指定します。

〈カラーコード〉には、〈パレット番号〉にどの色を対応づけるかを指定します。

パレットの個数およびカラーコードの値は、パレットモード(〔1〕COLOR 参照)によって異なります。各モードにおける〈パレット番号〉と〈カラーコード〉の関係は次のとおりです。

8色中・8色モード

0～7の〈パレット番号〉に0～7の8個の〈カラーコード〉を任意に指定します。

4096色中・8色モード

0～7の〈パレット番号〉に&H000～&HFFFの4096個の〈カラーコード〉から任意に指定します。

4096色中・16色モード

0から15の〈パレット番号〉に&H000～&HFFFの4096個の〈カラーコード〉から任意に指定します。

どのモードにおいても、パレット番号とカラーコードは任意に対応させることができます。また、同じカラーコードを複数のパレットに対応づけてもかまいません。

等号(=)、〈パレット番号〉および〈カラーコード〉を省略して、“COLOR”とすると、パレットとカラーコードの関係を初期化することができます(DISKモードでのみ有効)。パレットとカラーコードの初期状態に関しては、〔1〕COLORを参照してください。

注意：各モードにおけるカラーコードの値と色の関係は BASIC ユーザーズマニュアルを参照してください。

参照：[1] COLOR, サンプルプログラム 15, 16

COLOR@

機能	テキスト画面に書かれた文字などに色や機能を設定します。
書式	COLOR@ (X1, Y1)–(X2, Y2) [, <ファンクションコード>]
文例	COLOR@ (0, 0)–(79, 4), 2

COLOR@は、テキスト画面のキャラクタ座標の2点(X1, Y1), (X2, Y2)を対角とする四角形の領域に書かれている、文字やグラフィックキャラクタに色をつけたり、ブリンクなどの機能を設定したりします。

(X1, Y1), (X2, Y2)は、必ずキャラクタ座標でなくてはなりません。

<ファンクションコード>は、CONSOLEによって設定されているモードによって持つ意味が異なります。この機能および指定の仕方は[1] COLORの<ファンクションコード>の場合とまったく同様ですのでそちらを参照してください。省略された場合は、7を値として採用します。

この命令は、テキスト画面に書かれている文字などに対して有効です。したがって何も書かれていない場合は、何の作用もありません。またこの命令を実行した領域の上に新しく文字などを書いた場合、書かれた文字はこの命令の影響を受けません。

なお、COLOR@で指定した領域内に白黒モードで描かれたグラフィックスがある場合は、グラフィックスの色もCOLOR@で指定した色に変わります。カラーモードで描かれたグラフィックスは、COLOR@の影響を受けません。

参照：CONSOLE, [1] COLOR, サンプルプログラム 33

COMMON (DISK モード)

機能	CHAIN が実行された際、メモリ上のプログラムから、実行の移されたプログラムに変数を引き渡します。
書式	COMMON <変数名> [, <変数名> ...]
文例	COMMON A, B, XY(), NA\$

COMMON は、CHAIN によってメモリ上のプログラムからディスク上のプログラムに実行を移す際、メモリ上のプログラムの変数を実行の移されたプログラムに引き渡します。なお、CHAIN で MERGE オプションが指定されたときには、メモリ上のプログラムとディスク上のプログラムとが連結された後のプログラムに変数を引き渡します。

したがって、COMMON は必ず CHAIN とあわせて使われます。また、プログラム内において、COMMON は CHAIN の前にある必要があります。

〈変数名〉は 1 行の許す範囲(255 バイト以内)で何個でも並べることができますが、1 つのプログラム中の COMMON で同じ変数名を指定してはなりません。また配列変数名は () を後ろにつけ加えて表現します。

すべての変数を引き渡したいのであれば、CHAIN の ALL オプションを使う方が便利です。

参照 : CHAIN, サンプルプログラム 1, 2

COM ON / OFF / STOP

機能 RS-232C 回線からの割り込みの許可、禁止、停止を制御します。

書式 1) COM [(〈回線番号〉)] ON
2) COM [(〈回線番号〉)] OFF
3) COM [(〈回線番号〉)] STOP

文例 COM ON
COM(2) ON

RS-232C 回線に外部からの通信が入ったことによる割り込みの許可、禁止、停止を宣言します。

〈回線番号〉に指定できる値と意味は次のとおりです。(〈回線番号〉)を省略(カッコも含む)した場合は、第 1 回線となります。

- 1 : RS-232C 第 1 回線
- 2 : RS-232C 第 2 回線
- 3 : RS-232C 第 3 回線

ただし、2, 3 は専用のインタフェースボードが必要です。

書式1) 割り込みを許可します。以後、〈回線番号〉で指定した RS-232C 回線に受信が入るごとに割り込みが発生し、ON COM GOSUB によって定義された処理ルーチンに分岐します。

書式2) 割り込みを禁止します。以後、通信があっても処理ルーチンへの分岐は起こりません。

書式3) 割り込みを停止します。以後、通信があってもそのことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後、COM ON によって割り込みが許可されると、前に通信があったことによって、処理ルーチンに分岐します。

注意：プログラムの終了時には COM OFF を実行しておいてください。COM ON は、RS-232C 回線を OPEN した後実行されねば有効となりません。

なお、割り込み処理ルーチンに制御が移ると自動的に割り込み停止状態となりますので、割り込み処理ルーチンの先頭では COM OFF あるいは COM STOP を実行しないでください。

参照：ON COM GOSUB

CONSOLE

機能 テキスト画面モードの設定を行います。

書式 CONSOLE [<スクロール開始行>] [<スクロール行数>] [<ファンクションキー表示スイッチ>] [<カラー／白黒スイッチ>]

文例 CONSOLE 0, 24, 0, 1

CONSOLE ,, 1, 0

<スクロール開始行> で指定した行以降の、<スクロール行数> で指定した行数分を、画面上でスクロールする領域(スクロールウィンドウ)として設定します。画面のクリア(CLS あるいは PRINT CHR\$(12))は、このスクロールウィンドウに対して実行されます。

<ファンクションキー表示スイッチ>に 1 を指定すると、画面最下行にファンクションキーに定義されている文字列を表示します。0 を指定すると、表示をとりやめます。起動直後は表示される状態になっています。

<カラー／白黒スイッチ>に 1 を指定すると、テキスト画面をカラーモードにし、0 を指定すると白黒モードにします。起動直後は白黒モードになっています。

参照：[1] COLOR, COLOR@, サンプルプログラム 33

CONT

機 能 **STOP** キーあるいは **CTRL** + **C** の入力、または **STOP** によって停止したプログラムの実行を再開します。

書 式 **CONT**

文 例 **CONT**

CONT は通常デバッグのために用いられます。 **STOP** キー (または **CTRL** + **C**) の入力や、プログラム中の **STOP** の実行によってプログラムの実行を停止すると、ダイレクトモードで停止時における変数の内容などを調べることができます。この後、**CONT** を実行することにより、プログラムの実行を再開することができます。

注意：実行停止中にプログラム内容の変更を行った場合には、**CONT** による継続実行はできません。また、実行停止のタイミングによってはプログラムの継続実行ができない場合があります。

COPY

機 能 画面情報のハードコピーを行います。

書 式 **COPY** [〈機能〉]

文 例 **COPY 2**

COPY

2 種類の画面ハードコピー機能が用意されています。

(1) ROM モード BASIC の場合あるいは DISK モード BASIC でメモリスイッチ SW6 の 2'ピットが OFF の場合。

〈機能〉に指定できる値は 1～5 で、それぞれ次のように働きます。〈機能〉が省略された場合、3 を値として採用します。

- 1 : テキスト画面のみを出力。
- 2 : グラフィック画面のみを出力。
- 3 : テキスト画面とグラフィック画面を重ね合わせて出力。このときテキスト画面の情報はプリンタの印字体で出力される。
- 4 : グラフィック画面のみを出力。640×200 モードの場合に有効。
- 5 : テキスト画面とグラフィック画面を重ね合わせ、さらに縦方向に縮小して出力。

注意：PC-PR201 系プリンタを使用する場合は、メモリスイッチ SW5 の 2⁰ビットを ON にしてください。

(2) DISK モード BASIC で、メモリスイッチ SW6 の 2⁴ビットが ON の場合。

〈機能〉に指定できる値は 1～5 で、それぞれ次のように働きます。〈機能〉が省略された場合、3 を値として採用します。

なお、4、5 の機能はカラー出力の場合のみ有効です。

- 1 : テキスト画面のみを出力。
- 2 : グラフィック画面のみを出力(カラー出力の場合、白黒が反転される)。
- 3 : テキスト画面とグラフィック画面を合成して出力(カラー出力の場合、白黒が反転される)。
- 4 : グラフィック画面のみを出力(白黒は反転されない)(カラー出力用)。
- 5 : テキスト画面とグラフィック画面を合成して出力(白黒は反転されない)(カラー出力用)。

PC-PR601 系プリンタを使用する場合、上記の 1～3 の機能に対して、印字方向や印字倍率を指定することができます。詳しくは BASIC ユーザーズマニュアルを参照してください。

注意：PC-PR201 系または PC-PR601 系プリンタを使用する場合は、メモリスイッチ SW5 の 2⁰ ビットを ON にしてください。

また、カラー出力を行う場合、次の準備が必要です。

- PC-PR201V 系カラープリンタを接続する
- メモリスイッチ SW6 の 2³ビットを ON にする

詳しくは BASIC ユーザーズマニュアルを参照してください。

COS

関 数

機 能	余弦(コサイン)を得ます。
書 式	COS(〈数式〉)
文 例	X=RADIUS * COS(ANGLE) PRINT COS(3.14159/4)

〈数式〉の値に対する余弦値を得ます。〈数式〉の単位はラジアンで指定します。

〈数式〉に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：ATN, SIN, TAN, サンプルプログラム 6, 30

CSNG

関 数

機 能 整数値, 倍精度実数値を, 単精度実数値に変換します。

書 式 CSNG(〈数式〉)

文 例 A!=CSNG(B#/4)
PRINT CSNG(128%)

〈数式〉の値を有効数字 6 桁の単精度実数値に変換した値を得ます。結果の値が $-1.70141E+38$ ~ $1.70141E+38$ の範囲にない場合は, “Overflow” エラーになります。

参照：CINT, CDBL

CSRLIN

関 数

機 能 現在のカーソルの行位置を得ます。

書 式 CSRLIN

文 例 Y=CSRLIN

現在のカーソルの行(垂直)位置をキャラクタ座標で得ます。値の範囲は 25 行モードでは 0 ~ 24, 20 行モードでは 0 ~ 19 となります。得られる値は, 画面の最上行が 0, 最下行が 24 (または 19) を意味します。

参照：POS, サンプルプログラム 10

CVI / CVS / CVD

関 数

機 能 文字列を数値データに変換します。

書 式

- 1) CVI (<2 文字の文字列>)
- 2) CVS (<4 文字の文字列>)
- 3) CVD (<8 文字の文字列>)

文 例

```
A%=CVI(A$)
B=CVS("A3Bd")
C#=CVD(NUM$)
```

これらの関数は、MKI\$/MKS\$/MKD\$の各関数を用いてランダムデータファイルに書き込まれた文字型化数値データを、数値データにもどす際に使用します。

CVI : 2 文字(2 バイト)の文字列を整数値に変換(MKI\$の逆)。

CVS : 4 文字(4 バイト)の文字列を単精度実数値に変換(MKS\$の逆)。

CVD : 8 文字(8 バイト)の文字列を倍精度実数値に変換(MKD\$の逆)。

実際にランダムデータファイルから数値データを読み込む際には、まず GET でフィールド変数に文字型化数値データを読み込み、次にこれらの関数を用いて数値データに変換します。

参照 : MKI\$/MKS\$/MKD\$, サンプルプログラム 27

DATA

機 能 READ で読み込まれる数値定数、文字定数を定義します。

書 式 DATA <定数> [, <定数> ...]

文 例 DATA 1, CBA, 1465

<定数> には文字型か数値型のデータを指定します。READ で読み込む場合、DATA で指定した文字定数は文字変数に読み込まなくてはなりませんが、数値定数は文字変数、数値変数のいずれに読み込むこともできます。但し、定数式は使えません。

DATA 行中には、1 行(255 バイト)に入るだけのデータをセットすることができます。データはコンマ(,)によって区切りますが、文字定数で、その中にコンマやピリオド(.), 意味のある空白を置きたい場合、および日本語文字を含む文字列である場合は、その文字定数の前後をダブルクォーテーション(")でくくる必要があります。

1 つのプログラムには任意の数の DATA 行を置くことができます。DATA 行はプログラム

中のどこに置いてもかまいません。READ は行番号の小さい方から順番に、DATA 行中のデータを読み込んでいきます。

参照：READ, RESTORE, サンプルプログラム 3, 14

DATE\$

関 数

機 能 日付を得ます。

書 式 1) DATE\$
2) DATE\$="yy/mm/dd"

文 例 A\$=DATE\$
DATE\$="87/12/12"

DATE\$には常に現在の日付が"yy/mm/dd"(年/月/日)の形で入れられており、いつでもその内容を見ることができます。

なお、書式 2)を用いることにより、日付を変更することもできます。yy には 00～99, mm には 01～12, dd には 01～31 の範囲の整数値を指定します。yy, mm, dd は、それぞれ"/"(スラッシュ)で区切るようにしてください。

注意：日付は、バッテリーバックアップによって自動的に更新され、正しく維持されるようになっています。不用意に日付を変えないようにしてください。

参照：TIME\$

DEF FN

機 能 利用者定義関数を指定します。

書 式 DEF FN<名前>[(<パラメータリスト>)] = <関数の定義式>

文 例 DEF FNA(X, Y)=X * 2+Y * 3

DEF FN は、利用者が使うことのできる関数を定義する命令です。定義する関数は、数値関数、文字関数、その混在のいずれでもかまいません。

DEF FN は、プログラム中でしか実行することはできません。

<名前> は変数名として正しい形をしたものでなければなりません。

<パラメータリスト> には、<関数の定義式> の中で使われている変数とおなじ名前の変数を用います。この変数名は<関数の定義式> を評価する際にのみ有効なものであり、プログラム中に同一名の変数があってもかまいません。

〈関数の定義式〉は、その関数の演算内容を記述する式で、1 行の範囲(255 バイト以内)に限られます。

定義した関数は「FN 名前(変数)」という形で呼び出して使います。定義したときの変数は仮のものであるため、呼び出すときにはその時々で必要な変数に変えて指定することができます。ただし、変数の型は同一でなければなりません。

注意：DEF FN の〈関数の定義式〉中で使われている変数が、〈パラメータリスト〉内にはない場合は、その変数がその時点で持っている値が使われます。

DEF FN は、これによって定義される関数がプログラム中で呼ばれる前に実行されていなければなりません。

DEF FN は、DEF と FN の間に必ずスペースを入れなければなりません。スペースを省略して DEFFN とすると変数として扱われてしまいます。

参照：サンプルプログラム 39

DEFINT / DEFSNG / DEFDBL / DEFSTR

機 能 変数の型宣言を行います。

書 式

- 1) DEFINT 〈文字の範囲〉 [, 〈文字の範囲〉 ...]
- 2) DEFSNG 〈文字の範囲〉 [, 〈文字の範囲〉 ...]
- 3) DEFDBL 〈文字の範囲〉 [, 〈文字の範囲〉 ...]
- 4) DEFSTR 〈文字の範囲〉 [, 〈文字の範囲〉 ...]

文 例

```
DEFINT A, I-K
DEFSNG B
DEFDBL X-Z
DEFSTR L-N, Q
```

〈文字の範囲〉で指定された文字で始まる変数を、DEFINT では整数型に、DEFSNG では単精度実数型に、DEFDBL では倍精度実数型に、DEFSTR では文字型にそれぞれ定義します。

〈文字の範囲〉で指定できる文字は英字 1 文字で、複数個指定する場合はマイナス記号でつないでその範囲を示します。

注意：この命令によって行われる型宣言より、型宣言文字による指定(% , ! , # , \$)の方が優先されます。また、型宣言が行われていない文字で始まり、型宣言文字による指定もされていない変数は、すべて単精度変数とみなされます。

参照：サンプルプログラム 4

DEF SEG

機 能	セグメントベースを宣言します。
書 式	DEF SEG=〈セグメントベース〉
文 例	DEF SEG=&H9E00

BSAVE, BLOAD, PEEK, POKE などの命令でメモリのアドレスを指定する場合には、まず、アクセスしようとするメモリ空間を含むセグメントを、DEF SEG であらかじめ宣言する必要があります。そして、実際のメモリのアドレスの指定の際には、該当するセグメント内での相対アドレスで指定します。

〈セグメントベース〉には、セグメントの物理アドレスを 16 で割った値を指定します。

各変数や各配列が記憶されているセグメントベースは、VARPTR 関数により得ることができます。なお、配列については配列ごとに別セグメントとなっていますので、複数の配列の要素あるいは配列要素と変数にアクセスする場合には、そのつどセグメントベースを切り換えてアクセスすることが必要となります。

注意：DEF SEG は、DEF と SEG の間に必ずスペースを入れなければなりません。スペースを省略して DEFSEG とすると変数として扱われてしまいます。

参照：BLOAD, BSAVE, CLEAR, DEF USR, PEEK, POKE, VARPTR

DEF USR

機 能	USR で呼び出す機械語関数の番号と実行開始アドレスを定義します。
書 式	DEF USR[〈番号〉]=〈開始アドレス〉
文 例	DEF USR3=&HF000

〈番号〉は、0～9 までの値で、複数の機械語関数を用いる場合の識別を行います。最大 10 個までの機械語関数を準備、利用することができます。〈番号〉が省略された場合には 0 と解釈されますので、USR0 と USR は同じ意味となります。

〈開始アドレス〉は、〈番号〉によって指定される機械語関数の実行開始番地を、直前に実行された DEF SEG で指定されたセグメントベースからの相対アドレスで指定します。

なお、USR で呼び出したい機械語関数は、あらかじめメモリ上に用意しておかなくてはなりません。このためには、POKE や BLOAD を用いることができます。

DELETE

注意：DEF USR は、DEF と USR の間に必ずスペースを入れなければなりません。スペースを省略して DEFUSR とすると変数として扱われてしまいます。

参照：BLOAD, BSAVE, CLEAR, DEF SEG, POKE, USR

DELETE

機能 プログラムの部分削除を行います。

書式 DELETE <始点行番号> [- <終点行番号>]
 [<始点行番号>] - <終点行番号>

文例 DELETE 1050
DELETE *START-*SUB
DELETE .-500

<始点行番号> から <終点行番号> までのプログラムを削除します。<始点行番号> だけを指定した場合はその行だけを削除し、マイナス記号以下 <終点行番号> を指定した場合には、プログラムの先頭から指定行までを削除します。

なお、<始点行番号> および <終点行番号> の両方とも省略することはできません。

<行番号> にピリオド(.)を指定すると BASIC インタプリタ内のポインタが示している現在行を指定したことになります。実際には、LIST コマンドで最後に表示された行や、編集機能で最後に修正された行が現在行となります。

DIM

機能 配列変数の要素の大きさを指定し、メモリ領域に割り当てます。

書式 DIM <変数名> (<添字の最大値> [, <添字の最大値> …]) [, <変数名> (<添字の最大値> [, <添字の最大値> …])…]

文例 DIM A(12, 2), B\$(3)

DIM は、配列変数の添字の最大値を設定し、同時にメモリ上にその配列の領域を割り当てます。

<変数名> には、変数名として正しい形をしたものを指定します。

<添字の最大値> には、配列の要素を示す添字の、最大値を指定します。添字の最小値は、OPTION BASE によって、0 か 1 に指定することができます。実際に割り当てられる配列の要素の数は、OPTION BASE が 1 のときは<添字の最大値>に等しく、OPTION BASE が

0 のときは 〈添字の最大値〉+1 となります。

なお、OPTION BASE を指定しないで DIM を使用した場合、添字の最小値は 0 となります。
 〈添字の最大値〉を複数個指定すると、個数分の次元をもつ配列変数の指定となります。1 行
 中(255 バイト)に表記できる範囲であれば、次元数はいくつとってもかまいません。

ただし、配列の要素数および次元数は、メモリ容量の制限を受けます。配列変数のとるメモ
 リ領域が大きすぎると、“Out of memory” エラーとなります。

数値配列の場合、1 個の配列に対して指定できる要素の数は、整数型で 32767 個、単精度実
 数型で 16383 個、倍精度実数型で 8191 個までです。

DIM が実行された時点で、その配列のすべての要素の値は 0(文字型の場合はヌルストリン
 グ)に設定されます。

配列変数を消去するには、ERASE を用います。

注意：設定された最大値より大きな値の添字が用いられた場合は、“Subscript out of range”
 エラーが起こります。

DIM で宣言しなくとも配列変数を用いることができますが、その場合、添字の最大値は
 10 まで(OPTION BASE が 0 でも 1 でもかまわない)使用できます。

参照：ERASE、OPTION BASE サンプルプログラム 5

DRAW (DISK モード)

機 能	グラフィック描画サブコマンド列に従って、ワールド座標上で図形を描きます。
書 式	DRAW 〈文字列式〉
文 例	DRAW "D50R50U50L50" DRAW "C7T150, 50Z=TILE\$;"

DRAW では、複数のグラフィック描画サブコマンドを使って図形を描きます。〈文字列式〉
 内に指定可能なグラフィック描画サブコマンドは、一般的には次のような形式です。

〔〈修飾子〉〕〈識別子〉〔〈パラメータ 1〉〕〔, 〈パラメータ 2〉〕…

(1) 修飾子

〈修飾子〉としては、B および N を指定することができます。〈修飾子〉は〈識別子〉の補助
 的な役割を果たすもので、次のような意味をもちます。

B 〈識別子〉で示されるサブコマンドの実行は実際には行わず、LP(最終参照点)のみをサ
 ブコマンド実行後の値にします。

N 〈識別子〉で示されるサブコマンド実行後、LP を(0, 0)とします。

なお、B および N を同時に指定した場合は、〈識別子〉で示されるサブコマンドの実行は行われず、LP が(0, 0)となります。

(2) 識別子とパラメータ

〈識別子〉は、いろいろな描画機能を表すもので、英字1文字のサブコマンドで指定します。
〈パラメータ 1〉、〈パラメータ 2〉は、各サブコマンドに対応した数値定数または変数で、以下の形式で指定します。

〈数値定数〉[;]

または、

= 〈変数〉;

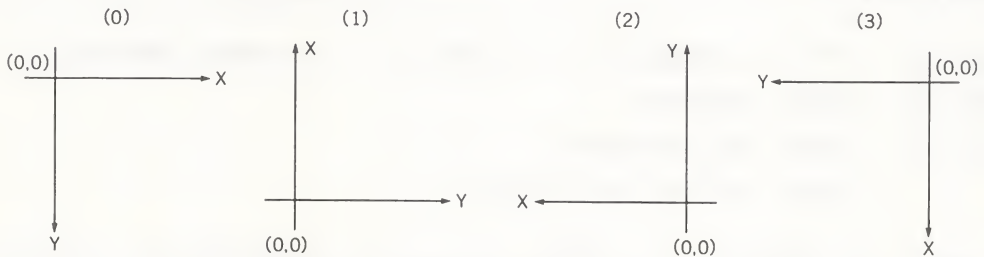
数値定数として指数形式を用いることはできません。また、16進表記で数値定数を表す場合には、その後に必ず、セミコロン(;)をつけなければなりません。

変数を指定する場合は、その直前には等号(=)を、直後にはセミコロン(;)を、それぞれ必ずつけるようにしてください。

〈識別子〉に指定するサブコマンドとその〈パラメータ〉について以下に説明します。

A 〈パラメータ 1〉

グラフィック描画サブコマンドの現座標系を、〈パラメータ 1〉に0～3までの整数値を指定することにより、以下の状態に規定します。



A サブコマンドによって規定された座標系は DRAW のグラフィック描画サブコマンドに対してのみ有効なもので、ワールド座標系が変化するわけではありません。

初期状態では、〈パラメータ 1〉の値は0となっています。

C 〈パラメータ 1〉

グラフィック描画サブコマンドによって描画される図形のパレット番号を規定します。〈パラメータ 1〉にはパレット番号を指定します。C サブコマンドによって規定されたパレット番号は、グラフィック描画サブコマンドに対して有効なものであり、フォアグラウンドカラーを変更するものではありません。

初期状態では、〈パラメータ 1〉の値はフォアグラウンドカラーとなっています。

D <パラメータ 1>

LP から、現座標系の Y 軸に沿って <パラメータ 1>×スケール値分だけ正の方向へ移動した点まで直線を描画します。スケール値については、S サブコマンドを参照してください。

E <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ正の方向に、さらに Y 軸に沿って <パラメータ 1>×スケール値分負の方向に移動した点まで直線を描画します。

F <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ負の方向に、さらに Y 軸に沿って <パラメータ 1>×スケール値分負の方向に移動した点まで直線を描画します。

G <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ負の方向に、さらに Y 軸に沿って <パラメータ 1>×スケール値分正の方向に移動した点まで直線を描画します。

H <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ正の方向に、さらに Y 軸に沿って <パラメータ 1>×スケール値分正の方向に移動した点まで直線を描画します。

L <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ負の方向へ移動した点まで直線を描画します。

M <パラメータ 1>, <パラメータ 2>

LP から、ワールド座標系の X 座標 = <パラメータ 1>, Y 座標 = <パラメータ 2> の点まで直線を描画します。

P [<パラメータ 1>]

LP を含み、<パラメータ 1>で指定したパレット番号で描かれた境界線で囲まれた領域を、色またはタイルストリングでぬりつぶします。<パラメータ 1>が省略された場合には、C サブコマンドで指定されているパレット番号が採用されます。P サブコマンドを実行するときは、LP はウィンドウ内になければなりません。P サブコマンドによってぬりつぶしを行う際の、色やタイルストリングの指定方法については、Z サブコマンドを参照してください。

Q <パラメータ 1>, <パラメータ 2>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分、Y 軸に沿って <パラメータ 2>×スケール値分それぞれ正の方向へ移動した点までを対角線とする長方形を描画します。

DRAW

R 〈パラメータ 1〉

LP から、現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分正の方向に移動した点まで直線を描画します。

S 〈パラメータ 1〉

グラフィック描画サブコマンドで指定される相対位置のパラメータのスケール値(倍率)を規定します。〈パラメータ 1〉はスケール値を表し、0 より大きな値でなければなりません。

初期状態では、〈パラメータ 1〉の値は 1 となっています。

T 〈パラメータ 1〉, 〈パラメータ 2〉

Q サブコマンドと同様にして長方形を描画した後、その内部をぬりつぶします。T サブコマンドによってぬりつぶしを行う際の、色やタイルSTRINGの指定方法については、Z サブコマンドを参照してください。

U 〈パラメータ 1〉

LP から、現座標系の Y 軸に沿って、〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します。

W 〈パラメータ 1〉, 〈パラメータ 2〉

LP から、現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分、Y 軸に沿って〈パラメータ 2〉×スケール値分それぞれ正の方向へ移動した点まで直線を描画します。

X 〈パラメータ 1〉

〈パラメータ 1〉で指定される文字変数内のグラフィック描画サブコマンド列に従い図形を描画します。文字列変数内に、再び X サブコマンドを含めることも可能です。なお、〈パラメータ 1〉には、必ず”= 〈文字変数〉;”の形を使用してください。

Y 〈パラメータ 1〉

D, E, F, G, H, L, M, Q, R, U および W サブコマンドで描画する直線または長方形のラインスタイルを規定します。〈パラメータ 1〉は、&H0~&HFFFF の範囲の整数でなければなりません。ラインスタイルの内容については、LINE を参照してください。

初期状態では、〈パラメータ 1〉の値は&HFFFF となっています。

Z 〈パラメータ 1〉

T および P サブコマンドでぬりつぶす際の、ぬりつぶしの色またはタイルSTRINGによる模様を規定します。〈パラメータ 1〉は、パレット番号またはタイルSTRINGが格納されている文字列変数でなければなりません。ただし、タイルSTRINGを指定する場合は、必ず”=〈文字変数〉;”の形を使用してください。タイルSTRINGについては、[2]PAINT を参照してください。

初期状態では、C サブコマンドにより規定されるパレット番号(C サブコマンドを実行していないときはフォアグラウンドカラー)となっています。

D, E, F, G, H, L, M, Q, R, T, U および W サブコマンドの実行では、修飾子として N が指定されていない限り、LP は指定された点に移ります。A, C, P, S, Y および Z サブコマンドの実行では LP は変化しません。

また、A, C, S, Y および Z サブコマンドの指定は、指定のある DRAW だけでなく、以降の DRAW にもその指定が引き継がれます。これらの指定は、SCREEN を実行すると初期状態にもどります。

参照 : LINE, [2] PAINT, サンプルプログラム 17

DSKF (DISK モード)

関 数

機 能	ディスクに関する情報を得ます。
書 式	DSKF(<ドライブ番号> [, <機能>])
文 例	PRINT DSKF(0) SECTORS=DSKF(2, 1)

<ドライブ番号> で指定されたディスクに関する情報を得ます。
<機能> は値として得ようとする情報の種類を決定するもので、次のように指定します。

- 省略時 : ディスクの残り容量(クラスタ単位)
- 0 : 最大トラック番号(=片面当りのトラック数-1)
 - 1 : 1 トラック当りのセクタ数
 - 2 : ディスクのサーフェイス(面)数-1
 - 3 : 1 トラック当りのクラスタ数 または 1 クラスタ当りのトラック数
フロッピーディスク(サーフェイス数-1=1)の場合: 1 トラック当りのクラスタ数
固定ディスク(サーフェイス数-1>1)の場合: 1 クラスタ当りのトラック数
 - 4 : ボリューム当りのクラスタ数
 - 5 : ディレクトリトラック番号
 - 6 : 1 クラスタ当りのセクタ数
 - 7 : FAT の開始セクタ番号
 - 8 : FAT の終了セクタ番号
 - 9 : FAT の数
 - 10 : ディスク属性の入っているセクタ番号

ドライブの種類やディレクトリ、FAT などについては、BASIC ユーザーズマニュアルを参照してください。

参照：DSKI\$, DSKO\$

DSKI\$ (DISK モード)

関 数

機 能	ディスクから直接、データを読み出します。
書 式	DSKI\$(<ドライブ番号>, <ヘッド番号>, <トラック番号>, <セクタ番号>)
文 例	D\$=DSKI\$(2, 1, 19, 1)

通常のファイル操作(OPEN, CLOSE など)とは別に、ディスク上の指定したセクタから直接データを読み出します。セクタ上に書かれているデータ 256 バイトをシステム用バッファに読み出すとともに、最初の 256 バイトの文字列を関数の値として返します。

文字列の長さは 255 バイトまでしか許されないために、関数の値として 256 バイトのデータの全てを得ることはできません。そこで、通常のランダムアクセスの場合と同じように FIELD によってシステム用バッファ(ファイル番号#0)へフィールド変数を割り付けることにより、これを解決することができます。あるいは、VARPTR 関数により、システム用バッファの置かれているメモリ番地を調べた上で、PEEK 関数によりバッファからデータを読み出すこともできます。

なお、システム用バッファは、ファイル番号#0 としてあらかじめ用意されているもので、OPEN によって開く必要はありません。

<ヘッド番号>, <トラック番号>, <セクタ番号>として指定できる値の範囲は、指定された<ドライブ番号>のディスクドライブの種類によって異なりますが、BASIC は、これらの値の範囲を自動的に調べ、不当であった場合には“Bad track/sector” エラーとします。

詳しくは BASIC ユーザーズマニュアルを参照してください。

参照：DSKF, DSKO\$, FIELD, VARPTR

DSKO\$ (DISK モード)

機 能	ディスクに対して直接書き込みを行います。
書 式	DSKO\$ <ドライブ番号>, <ヘッド番号>, <トラック番号>, <セクタ番号>
文 例	DSKO\$ 1, 0, 19, 1

通常のファイル操作とは別に、ディスク上の指定したセクタにシステム用バッファの 256 バイトのデータを直接書き込みます。#0 のシステム用バッファは、通常のディスクファイル操作の際には用いることはできませんが、DSKI\$ と DSKO\$ による直接操作の際にのみ、ユーザーが用いることができます。

書き込むデータの準備は、FIELD でファイル番号に #0 を指定することによって (OPEN は不要) システム用バッファへフィールド変数を割り付けた後、LSET, RSET により行います。あるいは、VARPTR 関数により、システム用バッファの置かれているメモリ番地を調べた上で、POKE によりバッファ中にデータを準備することもできます。

〈ヘッド番号〉、〈トラック番号〉、〈セクタ番号〉として指定できる値の範囲は、指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが、BASIC は、これらの値の範囲を自動的に調べ、不当であった場合には “Bad track/sector” エラーとします。

注意：DSKO\$ は、既存のディスクファイルを壊すおそれがありますので、ディスクおよびファイルの構成を正しく理解した上で用いてください。それぞれのディスクの諸元については DSKF 関数により調べることができます。詳しくは BASIC ユーザーズマニュアルを参照してください。

参照：DSKF, DSKI\$, FIELD, VARPTR

EDIT

機能 指定された行を画面上に表示し、以降 ROLL
UP キー、ROLL
DOWN キーなどによるプログラム編集を可能にします。

書式 EDIT 〈行番号〉

文例 EDIT 30

EDIT .

EDIT を実行すると〈行番号〉で指定された行が表示され、カーソルは行の先頭に移動します。以降、ROLL
UP キー、ROLL
DOWN キー、カーソルキーを始めとする、各種プログラム編集キーを使ったプログラムの編集が可能になります。

〈行番号〉にピリオド(.)を指定すると BASIC インタプリタ内のポインタが示している現在行を指定したことになります。実際には、LIST コマンドで最後に表示された行や、編集機能で最後に修正された行が現在行となります。

注意：P オプションつきでセーブされたファイルをロードして EDIT コマンドにより、内容を表示、変更しようとした場合には、“Illegal function call” エラーとなります。

END

機 能 プログラムの終了を宣言します。

書 式 END

文 例 END

プログラムの実行を終了し、すべてのファイルをクローズしてコマンドレベルにもどります。END はプログラムの実行を終了させたい所に置けばよく、また、いくつ置いておかまいません。

注意：プログラムの最後の END は省略することができますが、この場合、ファイルのクローズは行われません。

EOF

関 数

機 能 ファイルの終了コードを調べます。

書 式 EOF(<ファイル番号>)

文 例 IF EOF(1) THEN CLOSE 1 : END

<ファイル番号>で指定されたファイルが終わりに達したかどうかを調べる関数です。終わりに達していれば-1(真)、そうでなければ 0(偽)を返します。<ファイル番号>で指定されたファイルは、入力モードでオープンされていなければなりません。

指定されたファイルが RS-232C 回線(COM:)であった場合には、EOF 関数は、バッファが空であったときに値を真とします。

参照：サンプルプログラム 28

ERASE

機 能 配列変数を消去します。

書 式 ERASE <配列変数名> [, <配列変数名> ...]

文 例 ERASE C, D\$

<配列変数名>には、消去したい配列の変数名を指定します。添字や()は不要です。

ERASE を実行すると指定した配列変数が消去されますから、元の配列変数に割り当てられていた分だけメモリエリアが増加します。したがって、同一変数名の配列変数を DIM で新たに宣言したり、他の目的に使用することもできるようになります。

注意：ERASE を実行せずに同一変数名で配列変数を宣言しようとする、"Duplicate Definition" エラーが起こります。

参照：DIM, OPTION BASE, サンプルプログラム 5

ERL/ERR

関 数

機 能 エラーの発生した行番号および発生したエラーのエラーコードを保持しています。

書 式 1) ERL
2) ERR

文 例 IF ERL=100 THEN CLS : RESUME
IF ERR=7 THEN STOP

エラーが発生した時点で、ERL はエラーの発生した行番号を、ERR はエラーコードを保持しています。

一般に、ERR および ERL は ON ERROR GOTO によって指定したエラー処理ルーチンにおいて、独自のエラー処理を行うために使用します。

参照：ERROR, ON ERROR GOTO, サンプルプログラム 22

ERROR

機 能 エラー発生シミュレート、エラーコードのユーザー定義を行います。

書 式 ERROR <整数表記>

文 例 ERROR 200
ERROR A%

<整数表記> には、0 から 255 までの範囲の数を指定します。

この値が、すでに BASIC でエラーコードとして使われている場合は、エラーメッセージを表示するとともに、プログラムの実行を停止します。エラーコードが定義されていない場合は、プログラムの実行を停止します。

どちらの場合も、エラー発生時のプログラム行番号は ERL に、エラーコードは ERR にセットされます。したがって、ON ERROR GOTO によるエラー処理ルーチンを利用して、独自のエラー処理を行うことができます。

参照：ERL/ERR, ON ERROR GOTO, 付録 A. エラーメッセージとその対策, サンプルプログラム 22

EXP

関 数

機 能 e(自然対数の底)に対する指数関数の値を得ます。

書 式 EXP(<数式>)

文 例 E=EXP(1)

PRINT EXP(A/2)

<数式> の、e に対する指数演算の結果を値として得ます。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

FIELD(DISK モード)

機 能 ランダムファイルバッファにフィールド変数を割り当てます。

書 式 FIELD [#] <ファイル番号>, <フィールド幅> AS <文字変数> [, <フィールド幅> AS <文字変数> ...]

文 例 FIELD #1, 128 AS A\$, 64 AS B\$, 64 AS C\$

FIELD は、OPEN で設定されたランダムファイルバッファに対して、入出力を行うためのフィールドの名前、およびその長さを割り当てます。したがって、FIELD の実行にさきだって、OPEN によりファイルをランダムモードでオープンしておかなくてはなりません。

<文字変数>には、割り当てたいフィールドの名前を文字型の変数名として指定します。この変数は、バッファにデータを代入したり、代入したデータを参照するのに用いられ、フィールド変数と呼ばれます。

<ファイル番号>は、OPEN によってファイルバッファをオープンしたときに指定した番号です。

<フィールド幅> は、<文字変数> で指定したフィールド変数に割り当てる文字数で、バイト数で指定します。1つのバッファには、文字数合計が256バイトまでならば、いくつのフィールド変数を割り当ててもかまいません。

フィールド変数へ代入する(LSET あるいは RSET で行う)データの長さは、フィールド幅を超えてはなりません。

数値データをフィールドにセットする場合、数値データの型に応じてMKI\$, MKS\$, MKD\$の各関数を用いて文字型化してから LSET/RSET を行います。このとき、整数型の場合は2バイト長、単精度実数型の場合は4バイト長、倍精度実数型の場合は8バイト長にそれぞれ変換されますので、<フィールド幅> もこれに合わせて設定する必要があります。

2バイト系日本語文字列をフィールドにセットする場合、日本語1文字あたり2バイト分の

フィールドを必要とし、また日本語文字列の前後にシフトコード (KI コードおよび KO コードそれぞれ 2 バイト分) が挿入されます。〈フィールド幅〉の設定を行うときにはこのことを考慮するようにしてください。たとえば、

```
LSET A$="日本語"
```

のためには最低 10 バイト分のフィールドを必要としますので、フィールド幅には 10 以上を設定しなくてはなりません。

なお、1 つのバッファに対して、異なった形式で複数の FIELD を実行してもかまいません。

注意：各フィールド変数への値の代入を通常の代入文あるいは INPUT などで行わないでください。もし行くと、その変数は一般の文字変数領域に登録されてしまうため、FIELD でのバッファ割り当てが無効となり、正しいファイルの入出力が行われなくなります。

参照：GET, LSET / RSET, MKI\$ / MKS\$ / MKD\$, OPEN, PUT, サンプルプログラム 1, 2, 26, 27, 29

FILES / LFILES (DISK モード)

機能 ディスクに入っているファイルの名前、種類、大きさを出力します。

書式 1) FILES [〈ドライブ番号〉]
2) LFILES [〈ドライブ番号〉]

文例 1) FILES 2
2) LFILES

〈ドライブ番号〉で指定されたディスク上のファイルの名前、種類、大きさを表示します。書式 1) の場合ディスプレイに、書式 2) の場合プリンタに出力します。ファイルの大きさは、クラスタという単位で表示されます (クラスタについては、BASIC ユーザーズマニュアル参照)。〈ドライブ番号〉が省略された場合には、ドライブ 1 が選択されます。

ファイルの種類は、出力されるファイルの名前と、拡張子との間の区切り文字によって以下のように表されます。

区切り記号が空白：アスキーセーブされたプログラムファイル。

または、OPEN によって作られた BASIC のデータファイル。

ピリオド(.)：バイナリセーブされたプログラムファイル

アスタリスク(*)：BSAVE された機械語ファイル

参照：BSAVE, SAVE

FIX

関 数

機 能 数値の整数部を得ます。

書 式 FIX(<数式>)

文 例 F=FIX(B/3)

<数式> の値の小数点以下を取り去った値を得ます。

FIX と INT の違いは、<数式> の値が負のときに、INT は <数式> の値を超えない最大の整数を返すのに対し、FIX はたんに小数点部分だけを取り去った値を返すことです。

参照：INT

FOR...TO...STEP~NEXT

機 能 FOR から NEXT までの区間中にある一連の命令を、繰り返して実行します。

書 式 FOR <変数名>= <初期値> TO <終値> [STEP <増分>]

}

NEXT [<変数名>] [, <変数名>]

文 例 FOR J=0 TO 100 STEP 2

FOR K=10 TO 0 STEP -1

}

NEXT K, J

FOR~NEXT ループ中(FOR と NEXT の間)に置かれた命令を、FOR 中で指定した条件に従って繰り返して実行します。

<変数名>で指定される変数(ループ変数と呼ぶ)は、整数型または単精度型でなくてはなりません。

<初期値> には、変数の初期値を設定します。<終値> には、変数の最終値を設定します。<増分> には、初期値と最終値との間の増分を指定します。

文例を例にとると、はじめ J=0 が設定され、FOR 以降の命令を実行します。プログラムの実行が NEXT まで来ると、J の値が増分の 2 だけ増やされて J=2 となり、再び、FOR 以下が実行されます。J=100 となるまで FOR と NEXT の間の処理が繰り返されます。

STEP <増分> が省略された場合には、<増分> は 1 とみなされます。

次の場合には、FOR~NEXT は実行されずに、NEXT の次へ実行が移ります。

(1) <増分> が正の値で、<初期値> が <終値> より大きい場合。

(2) <増分> が負の値で、<初期値> が <終値> より小さい場合。

ただし、<変数> には <初期値> が代入されます。

1つの FOR～NEXT の中にはもう1つの FOR～NEXT を置くことができます(入れ子構造、ネスト構造などと呼ぶ)。この場合、それぞれの <変数名> には別のものを使わなければなりません。また、このとき、1つの FOR～NEXT は完全に他の FOR～NEXT の内部になければなりません。

注意：FOR と NEXT は必ず1対1に対応していなければなりません。

また、FOR～NEXT ループ内へ外部から GOTO などジャンプして入ってきたり、逆にループ内から外部へジャンプしたりするようなプログラムは、その動作が保証されなくなります。

参照：サンプルプログラム 6

FPOS

関 数

機 能 ファイル中での物理的な現在位置を得ます。

書 式 FPOS(<ファイル番号>)

文 例 HEAD=FPOS(4)

<ファイル番号> によって指定されたファイルが、最後に読み書きした位置を得ます。

指定されたファイルがディスクファイルの場合は、最後に読み書きしたセクタの番号が得られます。この番号はヘッド 0、トラック 0、セクタ 1 を 0 番とした通し番号となります。

指定されたファイルがプリンタの場合には、プリンタのヘッドの位置が得られます。この値は LPOS が返す値と同一です。

参照：LOC, LPOS

FRE

関 数

機 能 メモリの未使用領域の大きさを得ます。

書 式 FRE(<機能>)

文 例 PRINT FRE(0)

BASIC の利用可能なメモリ領域のうちの、各種の未使用領域の大きさを関数値として得ます。

〈機能〉に指定する値(整数値)とそれぞれの場合に得られる関数値の意味は次のとおりです。

- 0 : 未使用変数領域のバイト数
(単純変数およびストリング領域として使用可能な領域の残りバイト数)
- 1 : 未使用テキスト領域のバイト数
(プログラムテキストを入れるための領域の残りバイト数)
- 2 : 未使用変数領域と未使用テキスト領域の合計バイト数
- 3 : 配列データセグメントの未使用領域のバイト数

〈機能〉が0または2のときは、FRE は、ストリング領域の整理のための“ちり集め”処理を必ず行いますので、関数が値を返すまでに時間がかかることがあります。

GET

機 能 ファイル中のデータをファイルバッファに読み込みます。

書 式 GET [#] <ファイル番号> [, <数値>]

文 例 GET #3, 5

GET 1

〈ファイル番号〉で指定されたファイル中のデータを、対応するバッファ中に読み込みます。

GET は、指定されたファイルがディスクファイルか、キーボードファイルかによってその動作が異なります。

(1) ディスクファイルの場合

ランダムファイルからのデータをランダムファイルバッファに読み込みます。指定されたディスクファイルはランダムモードでオープンされていなければなりません。

〈数値〉はファイルのレコード番号として解釈され、指定されたレコードがバッファに読み込まれます。レコード番号の最小値は1、最大値は65000です。〈数値〉が省略された場合には、直前に行われた、GET、PUT で指定されたレコードの次のレコードが読み込まれます。

なお、GET を行った後で、INPUT #, LINE INPUT #などのシーケンシャル・アクセスを行った場合には、バッファに入っているデータから入力が行われますので注意してください。

(2) キーボードファイル(KYBD:)の場合

キーボードから入力した文字をバッファ中に読み込みます。キーボードファイルは入力モードでオープンされていなければなりません。

〈数値〉は、キーボードからバッファに読み込む文字(バイト)数と解釈されます。0 から 255 までの値で指定します。〈数値〉が省略されたとき、および0 が指定されたときには 256 文字を

読み込みます。

GET はバッファへの読み込みを行うものです。読み込んだデータは FIELD で指定した変数（フィールド変数）で参照することができます。

参照：FIELD, OPEN, PUT, サンプルプログラム 2, 27, 29

GET@

機能 画面上のグラフィックパターンを配列変数に読み込みます。

書式 GET[@] (Sx1, Sy1) — (Sx2, Sy2) , <配列変数名> [<添字>]
STEP(x, y)

文例 GET (100, 50) — (130, 70), G%

画面上のスクリーン座標の 2 点 (Sx1, Sy1) と (Sx2, Sy2) を対角点とする四角形のグラフィックパターンを、<配列変数名> で指定された配列変数に読み込みます。

座標の指定は、ビューポート内に展開されるスクリーン座標を使用します (PUT@も同様)。2 つ目の座標指定には、STEP を使って相対座標による指定もできます。

<配列変数名>は、グラフィックデータを読み込むために用意する数値型の配列変数の名前です。

<添字>は、グラフィックパターンを配列変数に読み込む際に、どの要素から格納し始めるかを指定するものです。省略した場合は配列の最初から格納します。<添字>を使うことにより、1 つの配列変数に対して複数のグラフィックパターンを格納することができます。

GET@を実行する前に、DIM で配列変数の大きさを宣言しておかなければなりません。確保すべき配列変数の大きさは、1 つの配列に 1 つのパターンしか読み込まない場合、次の計算式で求めることができます。複数パターンの場合は、それぞれのパターンについて計算し、その合計分を確保します。

$$\text{必要なバイト数} = ((\text{横のドット数} + 7) \div 8) * \text{縦のドット数} * M + 4$$

ただし、白黒モードのとき ———— M=1

8 色カラーモードのとき ———— M=3

16 色カラーモードのとき ———— M=4

$$\text{添字の値} = \text{必要なバイト数} \div N + 1$$

N は配列変数の型によって変わります。

整数型配列 ———— N=2

単精度型配列 ———— N=4

倍精度型配列 ———— N=8

この計算式によって求めた“添字の値”を、DIM で配列変数を宣言するときに〈添字の最大値〉(DIM 参照)として指定します。この計算式で求められる添字の値は、1 次元配列で添字の最小値が 0 のとき (OPTION BASE 参照) のものです。一般に GET@ では 1 次元の配列変数を用います。

注意：この GET@ は PUT@ と密接な関係にあり、通常ペアで使用されます。また、GET@ 実行後、LP (最終参照点) は (Sx2, Sy2) に移されます。

参照：DIM, OPTION BASE, PUT@, サンプルプログラム 18

GOSUB

機 能 サブルーチンを呼び出します。

書 式 GOSUB 〈行番号〉

文 例 GOSUB *SUB1

GOSUB 550

サブルーチンとは、他から独立したプログラムで、RETURN で終わっているものをいいます。GOSUB は、指定した〈行番号〉から始まるサブルーチンを呼び出します。

なお、〈行番号〉の代わりにラベル名を使うこともできます。

1 つのサブルーチンの中から他のサブルーチンを呼び出すこともできます。これを、サブルーチンの多重化と呼びます。この多重化は、メモリのスタック領域 (CLEAR 参照) の容量が許す限り行うことができます。

注意：サブルーチンを多重化してスタック領域が足りなくなると、“Out of memory”エラーが発生します。

参照：CLEAR, RETURN, サンプルプログラム 7, 30, 31, 33

GOTO/GO TO

機 能 指定された行へジャンプし、そこからプログラムを実行します。

書 式 1) GOTO 〈行番号〉

2) GO TO 〈行番号〉

文 例 GOTO 500

GOTO *EXIT

〈行番号〉の行へジャンプして、プログラムの実行を継続します。

〈行番号〉の代わりにラベル名を使うこともできます。

注意：“GO”と“TO”の間にスペース1個を入れても、まったく同じ意味です。しかし2個以上の場合には、GOTOとは解釈されません。

参照：サンプルプログラム 7, 28, 29, 30

HELP ON / OFF / STOP

機能

HELPキーによる割り込みの許可、禁止、停止を制御します。

書式

1) HELP ON

2) HELP OFF

3) HELP STOP

文例

HELP OFF

HELPキーを押したことによる割り込みの許可、禁止、停止を宣言します。

書式1) 割り込みを許可します。以後HELPキーを押すごとに割り込みが発生し、ON HELP GOSUBによって定義された処理ルーチンに分岐します。

書式2) 割り込みを禁止します。以後HELPキーを押しても処理ルーチンへの分岐は起こりません(HELPキーは通常の動作になります)。

書式3) 割り込みを停止します。以後HELPキーを押しても押されたことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後、HELP ONによって割り込みが許可されると、前にHELPキーを押したことによって、処理ルーチンに分岐します。

参照：ON HELP GOSUB

HEX\$

関 数

機 能 10 進数を 16 進数に変換し、その文字列を得ます。

書 式 HEX\$(〈数式〉)

文 例 A\$=HEX\$(X)

PRINT HEX\$(255)

〈数式〉の値を 16 進数に変換して、その文字列を得る関数です。〈数式〉の値は、-32768 から 32767(または 0 から 65535)までの範囲になければなりません。〈数式〉の値と得られる文字列との対応は次のとおりです。

〈数式〉の値	得られる 16 進表記文字列
0～32767	: 0～7FFF
-32768～-1	: 8000～FFFF
32768～65535	: 8000～FFFF

参照 : OCT\$, VAL, サンプルプログラム 24

IF…THEN～ELSE／IF…GOTO～ELSE

機 能 論理式の条件判断を行います。

書 式

1) IF 〈論理式〉 THEN	〈文〉	[ELSE	〈文〉]
	〈行番号〉		〈行番号〉	
2) IF 〈論理式〉 GOTO	〈行番号〉	[ELSE	〈文〉]
			〈行番号〉	

文 例

```
IF A$="Y" THEN *START ELSE *EXIT
IF SS THEN 200
```

〈論理式〉の条件によってプログラムの実行を制御します。

すなわち、〈論理式〉が真(0 以外)ならば THEN あるいは GOTO 以降を実行します。〈論理式〉が偽(0)ならば ELSE 以降が実行します。もし、ELSE 以降が省略されている場合には、次の行が実行されます。

IF…THEN(GOTO)～ELSE において、ELSE に続けて別の IF を置いて多重にすることができます。ただし、多重にできるのは 1 行(255 バイト)に書ける範囲に限られます。

注意：〈論理式〉が変数または定数の場合には、値が 0 以外のときに THEN (または GOTO)以降が、0 のときに ELSE 以降あるいは次の行が実行されます。

参照：サンプルプログラム 8

INKEY\$

関 数

機 能	押されているキーの文字を得ます。
書 式	INKEY\$
文 例	IF INKEY\$="" THEN *WAIT

押されているキーの文字(1文字)を取り出します。もしキーが押されていなければ INKEY\$の値はヌルストリング(空の文字列)となります。

キーボードバッファにすでに入力済みの文字が入っている場合には、バッファの先頭の1文字を取り出します。

INKEY\$では、押されたキーは画面上には表示されません。また、INKEY\$は、INPUT や INPUT\$などとは違い、キー入力待ちを行いません。

注意：CTRL + C, STOP キーは INKEY\$で取り出すことはできません。

参照：サンプルプログラム 29

INP

関 数

機 能	入力ポートから値を読み取ります。
書 式	INP(〈ポート番号〉)
文 例	A=INP(15)

〈ポート番号〉で指定された入力ポートから 8 ビットのデータを読み取り、それを関数値とします。〈ポート番号〉には 0～32767(&H0000～&H7FFF)の値を指定します。

〈ポート番号〉と装置の対応は、ハードウェアマニュアルを参照してください。

参照：OUT

INPUT

機能 キーボードから入力されたデータを、変数に代入します。

書式 INPUT [〈プロンプト文〉 ;] 〈変数〉 [, 〈変数〉 …]

文例 INPUT A

INPUT "住所"; B\$

INPUT が実行されると、プログラムはキーボードからの入力待ちの状態となります。

〈プロンプト文〉には、データの入力を受ける前に画面に表示されるメッセージを指定します。〈プロンプト文〉にセミコロン(;)が続いた場合には、さらにクエッションマーク(?)と1桁のスペースが画面に表示されます。コンマ(,)が続いた場合には〈プロンプト文〉の後には何も表示されません。

〈変数〉には、キーボードから入力したデータが代入されます。〈変数〉をコンマ(,)で区切って複数個指定した場合には、入力するデータもコンマで区切って〈変数〉の数だけ入力しなければなりません。また、対応する〈変数〉とデータの型とは一致していなければなりません。

何も入力しないでリターンキーだけを押した場合には、0 やヌルストリング(空の文字列)が入力されたとみなされます。この方法で0 やヌルストリングの入力ができますが、この場合も〈変数〉が複数の場合、必要数のコンマだけは入力しなければなりません。

文字変数に文字列を代入する場合、コンマや文字列の前後に意味のある空白を入力するには、ダブルクォーテーション(")で文字列を囲む必要があります。この場合ダブルクォーテーションは、文字として入力されません。これ以外の場合には、文字列をダブルクォーテーションで囲む必要はありません。

入力を中断したい場合、**STOP** キーあるいは **CTRL** + **C** キーを入力します。プログラムの実行中に入力を中断すると、**STOP ON** の状態にない限り、プログラムの実行も中断され、コマンドモードにもどります。このとき、それまでに入力した値はすべて無効となります(変数には値が代入されません)。

日本語の入力を行う場合には、**CTRL** + **XFER** を入力して日本語入力モードにします(BASIC ユーザーズマニュアル参照)。

注意: 〈変数〉と代入する値の型が違っていた場合には "? Redo from start" と画面表示して再び入力待ちとなります。

また、割り込み機能を使用しているプログラム中で、INPUT の実行中に割り込み動作が行われた場合、割り込み処理ルーチンから復帰したときに INPUT の次の命令に実行が移ってしまいますので、注意が必要です。

参照: INPUT WAIT, KINPUT, LINE INPUT, STOP ON/OFF/STOP, サンプルプログラム 1, 7, 8, 9, 22, 28, 29, 39

INPUT

機 能 シーケンシャルファイルからデータを読み込み、変数に代入します。

書 式 INPUT #〈ファイル番号〉, 〈変数名〉 [, 〈変数名〉 …]

文 例 INPUT # 1, A, B
INPUT # 2, NAMES\$

〈ファイル番号〉には、OPEN によって、そのファイルをオープンしたときに指定した番号を使います。

〈変数名〉の型は、ファイルから読み込まれるデータの型と対応していなければなりません。

ファイルに書き込まれているデータは、それが数値変数に代入される値(数字の並び)ならば、空白、コンマあるいはキャリッジリターン(CHR\$(13))で区切られていなければなりません。また、データが文字変数に代入される値(任意の文字の並び)ならば、コンマあるいはキャリッジリターンで区切られていなければなりません。

参照 : OPEN, サンプルプログラム 28

INPUT\$

関 数

機 能 指定されたファイルより指定された長さの文字列を読み込みます。

書 式 INPUT\$(〈文字数〉 [, [#] 〈ファイル番号〉))

文 例 WORD\$=INPUT\$(6, #2)

〈ファイル番号〉には、OPEN によってそのファイルをオープンしたときに指定した番号を使います。〈ファイル番号〉が省略された場合には、キーボードからの入力を読み込みますが、INPUT と異なり入力された文字は画面には表示されません。

〈文字数〉には、読み込みたい文字列の長さをバイト数で指定します。

INPUT\$は指定された〈文字数〉の文字が入力されるのを待ち続けますが、すでにバッファに入力済みのデータがある場合には、バッファ中から文字を読み込みます。

また、INPUT\$は、**STOP**キー、**CTRL**+**C**を除くすべての文字をそのまま読み込みますので、INPUT や LINE INPUT では入力することのできないキャリッジリターン(CHR\$(13))なども入力することができます。

参照 : INPUT, LINE INPUT, サンプルプログラム 22, 29

INT

関 数

機 能 小数点以下を切り捨てた整数値を得ます。

書 式 INT(<数式>)

文 例 PRINT INT(1.123)

A=INT(RND * 255)

<数式> の値を超えない最大の整数を得ます。

INT と FIX の違いは、<数式>の値が負のときに、FIX はたんに小数点部分だけを取り去った値を返すのに対し、INT は <数式> の値を超えない最大の整数を返すことです。

参照：FIX, CINT

JIS\$ (DISK モード)

関 数

機 能 2 バイト系日本語文字の漢字コードを得ます。

書 式 JIS\$(<文字列>)

文 例 C\$=JIS\$(KMID\$("漢字", 2, 1))

<文字列> の最初の 2 バイトを 16 進表記 4 桁の文字コードの文字列に変換します。

<文字列> の最初に 2 バイト系日本語文字を指定した場合には、日本語文字の前に KI コードが入っているため(画面には表示されない)、KI コード(&H1B4B)が得られます。したがって 2 バイト系日本語文字のコードを得るためには、KMID\$関数などを併用しなければなりません。

<文字列>の最初の 2 バイトが 1 バイト系文字 2 つの場合は、2 つの文字の 16 進数 2 桁のキャラクタコードが連続して得られます。

<文字列> が 2 バイト以下の場合は、“Illegal function call” エラーになります。

参照：KNJ\$, サンプルプログラム 36

KACNV\$ (DISK モード)

関 数

機 能 2 バイト系全角文字を、対応する 1 バイト系の英数カナ文字に変換します。

書 式 KACNV\$(<文字列>)

文 例 A\$=KACNV\$(B\$)

PRINT KACNV\$("アイウABC")

KEXT\$

〈文字列〉中の 2 バイト系全角文字を 1 バイト系英数カナ文字に変換します。なお、このとき 2 バイト系全角文字列中の KI/KO コードはすべて取り除かれます。

なお、〈文字列〉中に、対応する 1 バイト系英数カナ文字のない 2 バイト系全角文字が含まれていると、“Illegal function call” エラーとなります。

注意：〈文字列〉中の 2 バイト系半角文字は正しく変換されません。

参照：AKCNV\$, サンプルプログラム 35

KEXT\$ (DISK モード)

関 数

機 能 文字列の中から 1 バイト系英数カナ文字だけ、あるいは、2 バイト系日本語文字だけのどちらかを抜き出します。

書 式 KEXT\$(〈文字列〉, 〈機能〉)

文 例 A\$=KEXT\$(B\$, 0)
D\$=CHR\$(27)+"K"+KEXT\$("漢字 SAMPLE", 1)+CHR\$(27)+"H"

〈機能〉に指定する値により、特定のタイプの文字列だけを抜き出します。

- 0 : 1 バイト系英数カナ文字だけを抜き出す。
- 1 : 2 バイト系日本語文字だけを抜き出す。ただし、KI/KO コードは抜き出されません。

〈機能〉が 1 の場合、KI/KO コードが抜き出されませんので、得られた 2 バイト系日本語文字列を実際に使用する際には、その前後に KI/KO コードを付加しなくてはなりません。

指定したタイプの文字列がない場合、KEXT\$の値はヌルストリング(空の文字列)となります。

参照：サンプルプログラム 37

KEY

機 能 キーボードの上部にあるファンクションキーに文字列を定義します。

書 式 KEY 〈キー番号〉, 〈文字列〉

文 例 KEY 1, "Yes"+CHR\$(13)
KEY 3, CHR\$(&H22)+"BASIC"+CHR\$(&H22)

〈キー番号〉には、10 個あるファンクションキーの番号を指定します。たとえば、**f・1** キーならば 1 を、**f・2** キーならば 2 を指定します。

〈文字列〉には、それぞれのキーに記憶させておく文字列を指定します。

この命令を実行すると、以後、各ファンクションキーを押すたびに、それぞれに定義した文字列を入力できます。

各ファンクションキーには、最大 15 バイトまでの文字列およびコントロール文字を定義できます。キーボードから直接入力できない文字は、CHR\$関数を使って定義します。いちど定義したファンクションキーの内容は、新たに定義し直すかリセットするまで変わりません。

注意：ファンクションキーに 2 バイト系日本語文字列を定義することはできません。

参照：CHR\$, KEY LIST

KEY LIST

機能 ファンクションキーの内容を画面に表示します。

書式 KEY LIST

文例 KEY LIST

ファンクションキーの内容の一部は画面の最下行に、表示させておくことができますが、KEY LIST を使えば、そのすべての内容を画面に表示させることができます。

参照：CONSOLE, KEY

KEY ON/OFF/STOP

機能 ファンクションキーによる割り込みの許可、禁止、停止を定義します。

書式 1) KEY[(〈キー番号〉)] ON

2) KEY[(〈キー番号〉)] OFF

3) KEY[(〈キー番号〉)] STOP

文例 KEY ON

KEY(3) ON

〈キー番号〉には、1 から 10 までのファンクションキーの番号を指定します。(〈キー番号〉)を省略(カッコも含む)した場合には、すべてのファンクションキーを指定したことになります。

書式1) 割り込みを許可します。以後指定されたファンクションキーを押すごとに割り込みが発

KILL

生し、ON KEY GOSUB によって定義された処理ルーチンに分岐します。

書式2) 割り込みを禁止します。以後指定されたファンクションキーを押しても処理ルーチンへの分岐は起こりません(ファンクションキーの通常の動作になります)。

書式3) 割り込みを停止します。以後指定されたファンクションキーを押しても押されたことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後 KEY(n) ON によって割り込みが許可されると、前にファンクションキーを押したことが有効となり、処理ルーチンに分岐します。

注意：プログラムの終了時には KEY OFF を実行しておいてください。さもないと本来のファンクションキーの機能が失われます。

参照：ON KEY GOSUB、サンプルプログラム 30

KILL (DISK モード)

機能 ディスク上のファイルを削除します。

書式 KILL <ファイルディスクリプタ>

文例 KILL "2 : DATA1"

<ファイルディスクリプタ> で指定したファイルをディスクから削除します。

削除するファイルはクローズ状態になっていなければなりません。また、1 つの KILL では 1 つのファイルしか削除できません。KILL はすべてのディスクファイルについて使用できます。

注意：オープン状態のファイルを削除しようとするすると "File already open" エラーとなります。

KINPUT

機能 キーボードから入力された 2 バイト系日本語文字を、文字変数に代入します。

書式 KINPUT <変数名>

文例 KINPUT A\$

<変数名> には、文字変数を指定します。

KINPUT が実行されると、自動的に日本語入力モードに入り、キーボードからの入力待ちの状態になります。日本語の入力方法に関しては、BASIC ユーザーズマニュアルを参照してください。

注意：KINPUT は、他の INPUT 関連の命令と異なり、1つの文では1つの変数しか指定できません。KINPUT での入力はい数桁からになりますので、LOCATE で奇数桁を指定した場合は、その桁数に1を加えた桁が指定されたものと解釈されます。

KINSTR (DISK モード)

関 数

機 能 2バイト系日本語文字を含む文字列の中から指定文字列を捜して、その文字の位置を得ます。

書 式 KINSTR([<位置>], <文字列 1>, <文字列 2>)

文 例 PRINT KINSTR(3, A\$, "番号")
NUM=KINSTR("東京名古屋京都大阪", STA\$)

<文字列 1>の中から<文字列 2>が捜され、見つければその位置が、見つからなければ0が、それぞれ値として得られます。

<位置>には、<文字列 1>中で捜し始める位置を、文字数(整数値)で指定します。このとき、2バイト系日本語も1文字として、また、KI/KO コードも文字数に含めて数えます。<位置>が省略された場合は1が指定されたものとみなされ、<文字列 1>の最初から捜し始めます。

<文字列 2>に""(ヌルストリング)を指定すると、KINSTR の値は<位置>と同じになります。

KLEN (DISK モード)

関 数

機 能 2バイト系日本語文字を含む文字列中の、特定タイプの文字の合計数を得ます。

書 式 KLEN(<文字列> [, <機能>])

文 例 PRINT KLEN(A\$, 1)
LN=KLEN("東京 STATION")

<機能>に指定する値(整数値)により、特定のタイプの文字の合計数を得ます。

- 0 : 全体の文字数(2バイト系日本語文字も1文字として、KI/KO コードも1文字として数える)
- 1 : 1バイト系英数カナ文字の文字数
- 2 : 2バイト系日本語文字の文字数(KI/KO コードは含まない)
- 3 : 2バイト系全角文字の文字数
- 4 : 2バイト系半角文字の文字数
- 5 : KI/KO コードの数

〈機能〉を省略した場合は 0 と解釈されます。

なお、2 バイト系日本語文字のうち、全角の文字を 2 バイト系全角文字といい、半角のものを 2 バイト系半角文字といいます。

参照：サンプルプログラム 38

KMID\$ (DISK モード)

関 数

機 能 2 バイト系日本語文字を含む文字列の中から、任意の長さの文字列を抜き出します。

書 式 KMID\$(<文字列>, <式 1> [, <式 2>])

文 例 K\$=KMID\$(A\$, 4, 3)

PRINT KMID\$("JAPAN 日本 USA 米国", 1, 5)

〈文字列〉中の、〈式 1〉番目の文字から始まる〈式 2〉文字の長さの文字列を得ます。ここで、〈式 1〉、〈式 2〉に指定する文字数の単位は、1 バイト系英数カナ文字ならば 1 バイトを 1 文字として、2 バイト系日本語文字および KI/KO コードならば 2 バイトを 1 文字として、それぞれ数えます。

〈文字列〉の文字数が〈式 1〉より小さい場合、KMID\$の値はヌルストリング(空の文字列)となります。

〈式 2〉を省略した場合や、〈文字列〉中の〈式 1〉番目の文字から右の文字数が〈式 2〉より小さい場合は、〈文字列〉中の〈式 1〉番目の文字から始まる右のすべての文字列が結果として得られます。

参照：サンプルプログラム 35, 36

KNJ\$ (DISK モード)

関 数

機 能 漢字コード文字列 4 桁を 2 バイト系日本語文字 1 文字に変換します。

書 式 KNJ\$(<文字列>)

文 例 PRINT KNJ\$("1B4B") + KNJ\$("3441") + KNJ\$("1B48")

〈文字列〉に指定した漢字コード 4 桁を、2 バイト系日本語文字 1 文字に変換します。このとき、〈文字列〉の内容は次の条件をすべて満たしてはなりません。1 つでも満たされていないと、"Illegal function call" エラーになります。

- (1) 文字列が4桁以上であること
- (2) 最初の4桁の文字が、いずれも16進数(0～F)の範囲内の文字であること
- (3) 4桁の文字の組み合わせが、漢字コードの範囲内あるいはKI/KOコードであること

〈文字列〉が4桁を超える場合、最初の4桁のみが用いられます。

なお、得られた2バイト系日本語文字を実際に使用する際には、その前後にKI/KOコードを付加しなくてはなりません。

参照：JIS\$, サンプルプログラム 34, 36

KPLOAD (DISK モード)

機能 利用者定義文字パターンをシステムに登録します。

書式 KPLOAD 〈漢字コード〉, 〈整数型配列名〉

文例 KPLOAD &H762A, CHRPTN%

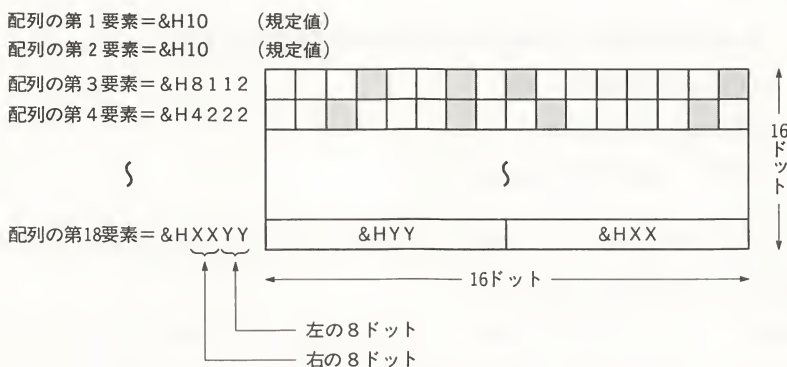
ユーザーが作字した利用者定義文字のパターンをシステムに登録する命令です。

〈漢字コード〉には登録するコードを指定します。指定可能な値の範囲は、&H7621～&H767E および&H7721～&H777E までです。この範囲にないときは、“Illegal function call”エラーとなります。

〈整数型配列名〉には文字パターンの格納された配列変数を指定します。〈整数型配列名〉に指定する配列変数は、あらかじめDIMにより、1次元18要素の整数型配列として宣言しておいてください。

配列内に格納されている文字パターンは次のような形式でなければなりません。

配列の第3要素から第18要素までに、次のように実際の文字パターンのドットイメージを格納してください。



参照：サンプルプログラム 34

KTYPE (DISK モード)

関 数

機 能 2 バイト系日本語文字を含む文字列中の、指定位置の文字のタイプを得ます。

書 式 KTYPE(〈文字列〉, 〈式〉)

文 例 B=KTYPE(A\$, 5)

PRINT KTYPE("日本語", 2)

〈文字列〉中の〈式〉バイト目の文字のタイプを得ます。

〈式〉には、タイプを得たい文字の位置を、〈文字列〉の先頭から数えたバイト数で指定します。この場合、1 バイト系英数カナ文字は1文字を1バイトと数えますが、その他の文字はすべて1文字を2バイトと数えます。

得られる値とタイプとの関係は次のとおりです。

得られる値 タイプ

- 0 : 1 バイト系英数カナ文字
- 1 : 2 バイト系全角文字
- 2 : 2 バイト系半角文字
- 3 : KI コード
- 4 : KO コード

〈式〉の値が0 または 〈文字列〉の文字数より大きい場合、あるいは〈文字列〉がマルチストリング(空の文字列)の場合は、"Illegal function call" エラーとなります。

参照：サンプルプログラム 38

LEFT\$

関 数

機 能 文字列の左側から任意の長さの文字列を抜き出します。

書 式 LEFT\$(〈文字列〉, 〈式〉)

文 例 B\$=LEFT\$(A\$, 4)

PRINT LEFT\$("standing", 5)

〈文字列〉の左側(最初)から〈式〉で指定した桁数の文字列を抜き出します。〈式〉の値は0 から 255 の範囲になければなりません。

〈式〉の値が0 ならば、LEFT\$の値はマルチストリング(空の文字列)となります。〈式〉が〈文字列〉の文字数より大きければ、LEFT\$の値は〈文字列〉とまったく同じになります。

参照：MID\$, RIGHT\$, サンプルプログラム 25

LEN

関 数

機 能 文字列の合計バイト数を得ます。

書 式 LEN(<文字列>)

文 例 PRINT LEN(A\$)

L=LEN("TEST")

<文字列> 全体の合計バイト数を得ます。

出力されない文字や空白も数えられます。ここで、出力されない文字とは、キャラクタコードで 0 から 31 までの、通常コントロールコードと呼ばれるものをいいます。

LEN では、<文字列> 中にどんなタイプ(1 バイト系, 2 バイト系)の文字が含まれていても、合計のバイト数が得られます。

参照：KLEN, サンプルプログラム 25

LET

機 能 変数に値を代入します。

書 式 [LET] <変数名>= <式>

文 例 LET I=I+1

I=(I+1)*J

A\$=STRING\$(20,"*")

LET はなくてもかまいません。

<式> の内容は、数値、文字列のいかなる型であってもかまいません。ただし、<変数名> と <式> の型は同じでなければなりません。型の異なる数値変数への代入では左辺の精度に合わせた代入が行われます。

注意：代入では必ず、左辺に変数を置かなければなりません。

LINE

機 能 指定した 2 点間に直線を描きます(パラメータ指定により四角形も描きます)。

書 式

```

LINE  [ (Wx1, Wy1) ] - [ (Wx2, Wy2) ] [ , <パレット番号 1> ]
      [ STEP(x1, y1) ] [ STEP(x2, y2) ]
      [ , [ B ] ] [ , <ラインスタイル> ]
      [ BF ] [ <パレット番号 2> ]
      [ <タイルストリング> ]

```

文 例

```

LINE(100, 100)-(135, 100), 7,, &HF99F
LINE-STEP(30, 30), 7, BF, 7

```

ワールド座標系の 2 点(Wx1, Wy1)と(Wx2, Wy2)を結ぶ直線を描きます。

(Wx1, Wy1)が省略された場合、LP(最終参照点)の値を採用します。STEP を使って相対座標で指定することもできます。

<パレット番号 1>には、描く線の色を指定します。省略された場合は、そのとき COLOR で設定されているグラフィック画面のフォアグラウンドカラーが採用されます。

次のパラメータには、B か BF のどちらかを指定することができます。B は Box の意味で、(Wx1, Wy1)と(Wx2, Wy2)の 2 点を対角とする四角形の箱を描きます。BF は Box Fill の意味で、描いた四角形の内部をぬりつぶします。

<ラインスタイル>は、描く線の形態を設定するパラメータであり、&H0 から&HFFFF は(16 進数)の値をとることができます。この値とラインの形態との関係は、図のとおりです。<ラインスタイル> が省略された場合は、直線を引きます。BF の指定時には指定できません。

<パレット番号 2>は、四角形の内部をぬりつぶす際の色を、<タイルストリング>は、ぬりつぶす際の模様を指定します。<パレット番号 2>および<タイルストリング>は、BF 指定のときだけ有効です。BF 指定があり、<パレット番号>、<タイルストリング>のいずれも省略された場合には、四角形を描いたのと同じ色で内部をぬりつぶします。

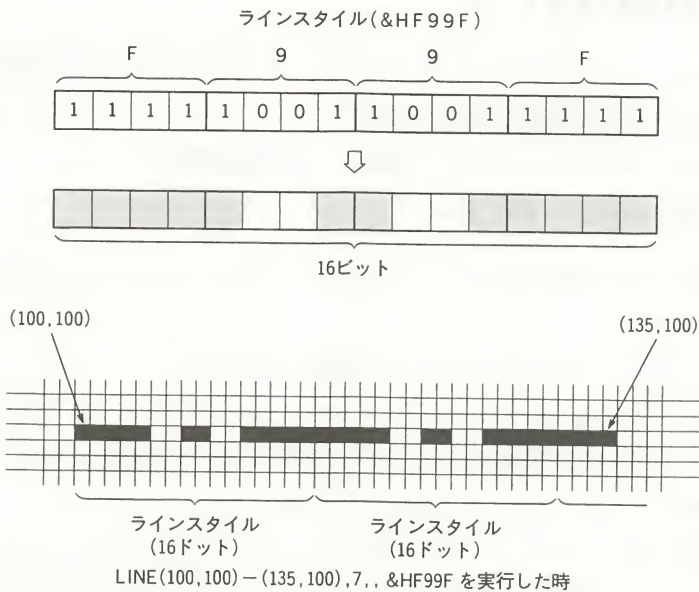
パレット番号については [2] COLOR を、<タイルストリング>については [2] PAINT を参照してください。

LINE を実行した場合、LP は(Wx2, Wy2)に移動します。

この図は文例を実行したものです。<ラインスタイル>の 16 進数の値を 2 進数 16 ビットで表したものと画面上の 16 ドットが対応しています。“1”のビットに対応するドットが表示され、“0”のビットに対応するドットは表示されません。この例では、水平座標の 100 から 135 までの 36 ドットの間に、1 点鎖線のラインスタイルが繰り返されています。線を 16 ドット以上引いた場合には、ラインスタイルが画面上の 16 ドットごとに繰り返されることになります。

このようにラインスタイルを指定すると、16 ドット単位で線の形態を決めることができます

から、折れ線グラフや図面の作成の際、点線や1点鎖線などを簡単に描くことができます。



参照：[2] COLOR, [2] PAINT, サンプルプログラム 15, 16, 17, 18, 20

LINE INPUT

機能 キーボードから入力されるデータ(255 バイト以内)を、区切ることなく一括して文字変数に代入します。

書式 LINE INPUT [<プロンプト文>;] <文字変数名>

文例 LINE INPUT "NAME"; NA\$
LINE INPUT B\$

LINE INPUT では、キーボードから入力されたすべてのデータがコンマなどの区切り記号も含めて<文字変数名>で指定された変数に代入されます(INPUT の場合は区切り記号で区切られたデータの1つ1つが順次抽出され、変数に代入されます)。

<プロンプト文>には、入力を受ける前に表示するメッセージを指定します。

入力を中断したい場合、**STOP** キーあるいは **CTRL** + **C** キーを入力します。プログラムの実行中に入力を中断すると、STOP ON の状態にない限り、プログラムの実行も中断され、コマンドモードにもどります。このとき、それまでに入力した値はすべて無効となります(文字変数には値は代入されません)。

参照：INPUT, LINE INPUT #, STOP ON/OFF/STOP

LINE INPUT

機能 シーケンシャルディスクファイルより、1行(255バイト以内)単位データを一括して文字型変数に読み込みます。

書式 LINE INPUT # <ファイル番号>, <文字型変数名>

文例 LINE INPUT #1, A\$

<ファイル番号>には、OPENによって、そのファイルをオープンしたときに指定した番号を使います。

<文字変数名>は、データが代入される文字変数の名前です。

LINE INPUT #は、シーケンシャルファイルの中から、キャリッジリターン(CR)コード+ラインフィード(LF)コードで区切られている1行単位を読み込みます。ここで、CRコードはCHR\$(13)、LFコードはCHR\$(10)に相当します。

LINE INPUT #は、データファイルのそれぞれの行がスペースやコンマによっていくつかの欄に分割されているときや、アスキーセーブしたプログラムを他のプログラムのデータとして読み込むときなどに特に有効です。

参照：INPUT #, OPEN

LINE INPUT WAIT

機能 キーボードから入力されるデータを変数に代入します。その際、入力待ち時間を制限することができます。

書式 LINE INPUT WAIT <待ち時間>, [<プロンプト文> ; |] <文字型変数名>

文例 LINE INPUT WAIT 50, "No.=", NO\$

LINE INPUT WAITは時間制限付きのLINE INPUTです。したがって、LINE INPUTとINPUT WAITの両方の機能を兼ねそなえています。詳しくはそれぞれの命令を参照してください。

参照：LINE INPUT, INPUT WAIT

LIST/LLIST

機能 メモリ上にあるプログラムの全部、または一部を表示あるいは印刷します。

書式 1)LIST [**<始点行番号>**][**- <終点行番号>**]
2)LLIST [**<始点行番号>**][**- <終点行番号>**]

文例 LIST .
LLIST 100-200

LIST を実行すると、プログラムの内容が、書式 1)は画面に、書式 2)はプリンタに、それぞれ出力されます。

パラメータの指定方法とリストの関係は次のとおりです。

- <始点行番号>-<終点行番号>** : その範囲のすべてのプログラム行をリストする。
- <始点行番号>-** : 始点行番号に始まりそれよりも大きい行番号のプログラム行すべてをリストする。
- <終点行番号>** : 最も若い行番号のプログラム行から終点行番号までをリストする。
- <始点行番号>** : その行番号のプログラム行だけをリストする。
- 行番号を省略** : プログラム行をすべてリストする。

行番号にピリオド(.)を指定すると、BASIC インタプリタ内のポインタが示している現在の行を指します(たとえば、最後に修正された行)。

リストの出力は**STOP** キーまたは**CTRL**+**C**の入力で終わります。

この命令は、実行し終るといつでもコマンドレベルにもどります。

LOAD

機能 プログラムをメモリにロードします。

書式 LOAD **<ファイルディスクリプタ>** [, R]

文例 LOAD "2 : DEMO", R
LOAD "COM : N82NN"

<ファイルディスクリプタ> には、メモリにロードするプログラムファイル(ディスク、RS-232C 回線、キーボード、カセット)を指定します。LOAD を実行すると、すべての開いているファイルは閉じられ、変数に代入されている値は失われます。

R オプションをつけると、ファイルは開いたままで、プログラムをロード後、直ちに実行を開始します。

LOAD?

注意：RS-232C 回線からのプログラムロードの場合、プログラムのロードが完了しても、自動的に LOAD コマンドは終了しません。STOP キーで強制的にブレーク(中止)しなければなりません。

LOAD ?

機能 カセットに正しくプログラムがセーブできたかどうかを確認します。

書式 LOAD? <ファイルディスクリプタ>

文例 LOAD? "CAS2 : TEST"

<ファイルディスクリプタ>により指定されたカセット上のプログラムの内容と、メモリ中のプログラムの内容が同一であるかを調べます。比較をするのみで実際のロード動作は行われません。両者が異なっている場合は、"Bad"と表示されます。このコマンドは、SAVE の直後に正しくセーブされたことを確認する目的で用います。

注意：LOAD? は、カセットファイルに対してのみ有効です。他の装置上のファイルが指定された場合には、LOAD とまったく同じ動作をし、メモリ中のプログラムが失われますので注意してください。

LOC

関数

機能 ファイル中での論理的な現在位置を得ます。

書式 LOC(<ファイル番号>)

文例 LAST=LOC(2)

<ファイル番号> で指定されたファイルの種類によって、得られる値の意味が異なります。

ランダムディスクファイルの場合

最後に読み書き (GET/PUT) を行ったレコードのレコード番号が得られます。

シーケンシャルディスクファイルの場合

オープンしてから現在までに読み書きしたレコード数が得られます。

キーボードファイルの場合

割り込みによって受け付けられ、キーボードバッファ中にたまっている文字数が得られます。

RS-232C 回線ファイルの場合

割り込みによって受け付けられ、入力バッファ中にたまっている文字数が得られます。
読み取り動作(INPUT #, INPUT\$など)が行われずに、割り込みによる RS-232C 回線からの受信が続くと、やがてバッファが一杯になりエラーとなるため、これを監視するのに LOC を使用することができます。

LOCATE

機 能	テキスト画面のカーソルを指定位置へ移動します。
書 式	LOCATE [$\langle X \rangle$] [, $\langle Y \rangle$] [, $\langle \text{カーソルスイッチ} \rangle$]
文 例	LOCATE 10, 10 LOCATE ,, 0

X は水平座標を表し、省略された場合は 0 とみなされます。Y は垂直座標を表し、省略された場合は現在カーソルがセットされている行とみなされます。これらは、ともに画面の左上を (0, 0) とするキャラクタ座標により指定します。

$\langle \text{カーソルスイッチ} \rangle$ はカーソルの表示状態を決めるスイッチで、0 を指定すると表示されなくなり、1 を指定すると表示されるようになります。

注意：水平座標が奇数桁の場合、日本語の表示あるいは日本語入力を正しく行うことはできません。

参照：サンプルプログラム 10, 33

LOF

関 数

機 能	ファイルの大きさを得ます。
書 式	LOF($\langle \text{ファイル番号} \rangle$)
文 例	MAXREC=LOF(2)

$\langle \text{ファイル番号} \rangle$ で指定されたファイルの種類によって、得られる値の意味が異なります。

ディスクファイルの場合

ファイルの大きさがセクタ数で得られます。この値は、ファイルがランダムファイルであれば、そのファイルの最大レコード番号に相当します。

RS-232C 回線ファイルの場合

入力バッファの残りのバイト数(入力可能な余裕バイト数)が得られます。

参照：サンプルプログラム 29

LOG

関 数

機能	自然対数を得ます。
書式	LOG(<数式>)
文例	PRINT LOG(35/9) LA=LOG(X * Y)

〈数式〉の、自然対数(eを底とした対数)を値として得ます。

〈数式〉に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：EXP

LPOS

関 数

機 能	現在のプリンタのヘッド位置を得ます。
書 式	LPOS (<式>)
文 例	HEAD=LPOS(0)

使用中のプリンタの、現在のヘッド位置(桁位置, すなわち水平位置)を得ます.

〈式〉は、ダミーであって意味を持ちませんが、省略できません。

参照: WIDTH LPRINT

LPRINT

機能	プリンタにデータを出力します。
書式	LPRINT [〈式〉] [, 〈式〉…] [, ; ;]
文例	LPRINT A\$, D2 LPRINT CHR\$(&H41) ; CHR\$(&H42) ; CHR\$(&H43)

指定した式の値や文字列などのデータを、プリンタに出力します。

LPRINT は、データをプリンタに出力する点が違うだけで、その動作は PRINT と同じです。

参照：PRINT

LPRINT USING

機 能 文字列、数値などのデータを編集し、プリンタに出力します。

書 式 LPRINT USING <書式制御文字列>;<式> [, | <式>…] [, |
; ;]

文 例 LPRINT USING "@ ###"; A\$, B
LPRINT USING "¥¥#.###"; C

<書式制御文字列>によって決定される領域や書式に従って、<式>に指定された各種データをプリンタに出力します。

LPRINT USING は、データをプリンタに出力する点が違うだけで、その動作は PRINT USING と同じです。

参照：PRINT USING

LSET / RSET (DISK モード)

機 能 ランダムファイルバッファのフィールドにデータを代入します。

書 式 1) LSET <文字変数>=<文字列>
2) RSET <文字変数>=<文字列>

文 例 LSET A\$="単価"
RSET B\$=MK\$\$(I)

<文字変数>には、FIELD でランダムファイルバッファ上に割り当てられたフィールド変数を指定します。

<文字列>を指定することにより、フィールド変数に対応するフィールドにデータ(文字型)を代入することができます。<文字列>の文字数が FIELD で割り当てられたフィールド変数の長さより短い場合、LSET では左詰め、RSET では右詰めでフィールドを満たします。また、<文字列>がフィールドより長い場合は、LSET、RSET とともに右側の部分が失われます。

注意：〈文字変数〉は、あらかじめ FIELD で定義されていなければなりません。定義されていない変数名を用いると、データを正しく代入することができません。

また、〈文字列〉に指定するデータは文字型でなければなりませんので、数値型データは MKI\$, MKS\$, MKD\$のいずれかにより文字型データに変換しておかなければなりません。

参照：FIELD, GET, MKI\$／MKS\$／MKD\$, PUT, サンプルプログラム 26, 29

MAP

関 数

機 能	スクリーン座標，ワールド座標の相互変換を行います。
書 式	MAP(〈数式〉, 〈機能〉)
文 例	SY1=MAP(WY1, 1)

〈数式〉で指定されるスクリーン座標あるいはワールド座標の値を、〈機能〉の指定に従い、対応するワールド座標あるいはスクリーン座標に変換します。

〈機能〉に指定する値により、次のような変換を行います。

- 0 ： 〈数式〉をワールド座標系における x 座標とみなし、これを対応するスクリーン座標系での x 座標に変換する ($W_x \rightarrow S_x$)。
- 1 ： 〈数式〉をワールド座標系における y 座標とみなし、これを対応するスクリーン座標系での y 座標に変換する ($W_y \rightarrow S_y$)。
- 2 ： 〈数式〉をスクリーン座標系における x 座標とみなし、これを対応するワールド座標系での x 座標に変換する ($S_x \rightarrow W_x$)。
- 3 ： 〈数式〉をスクリーン座標系における y 座標とみなし、これを対応するワールド座標系での y 座標に変換する ($S_y \rightarrow W_y$)。

MAPを用いれば、ワールド座標からスクリーン座標への変換ができますから、GETやPUTのようにスクリーン座標で位置を指定する操作の場合にも、ワールド座標を変換して指定することができます。

注意：MAPは、変換の結果がスクリーン座標系あるいはワールド座標系から外れてもエラーとなりません。

MERGE (DISK モード)

機 能 メモリ上のプログラムに、ディスク上のプログラムファイルを混合します。

書 式 MERGE <ファイルディスクリプタ>

文 例 MERGE "2 : TEST"

メモリ上のプログラムと、<ファイルディスクリプタ>により指定したディスク上のプログラムファイルを混合(マージ)して1つのプログラムにし、メモリ上に置きます。

メモリ上のプログラムと、ディスク上のプログラムファイルに同一の行番号があった場合には、ディスク上のプログラムファイル中の行が採用されます。

なお、マージ後、ただちにそのプログラムを実行させたいときは、MERGE オプション付きの CHAIN を使うことができます。

注意：混合するプログラムファイルはアスキーセーブされていなければなりません。そうでない場合には、"Sequential I/O only" エラーになります。

参照：CHAIN, SAVE

MID\$

機 能 文字列の一部を置き換えます。

書 式 MID\$(<文字変数>, <式 1> [, <式 2>]) = <文字列>

文 例 MID\$(A\$, 3) = "ABC"

MID\$(B\$, N, 2) = C\$

<文字変数>に代入されている文字列の<式 1>番目の文字から<式 2>個の文字を、<文字列>の最初から<式 2>個分の文字列で置き換えます。

<式 1> は 1 から 255 の範囲、また、<式 2> は 0 から 255 の範囲になければなりません。

<文字列> には文字変数を用いることもできます。

<式 2> が省略された場合、または<式 2>の値が<文字列>の文字数を超える場合、<文字変数>に代入されている文字列を、<文字列>のすべての文字と置き換えます。

注意：この命令は2バイト系日本語文字列にも利用できますが、その際には文字列の前後に入る KI/KO コードに注意してください。

また、1バイト系と2バイト系の混在文字列に使用する場合は、文字の境界に充分注意しないと、予期しない結果となります。

参照：MID\$(関数)

MID\$

関 数

機 能 文字列の中から任意の長さの文字列を抜き出します。

書 式 MID\$(〈文字列〉, 〈式 1〉 [, 〈式 2〉])

文 例 B\$=MID\$(A\$, 2, 3)
COUNTRY\$=MID\$("JPNUSAFRAGBRCAN", N * 3 + 1, 3)

〈文字列〉の〈式 1〉番目の文字から〈式 2〉桁の文字列を抜き出します。

〈式 1〉は 1 から 255 の範囲、また、〈式 2〉は 0 から 255 の範囲になければなりません。

〈式 2〉を省略した場合や、〈文字列〉の〈式 1〉番目の文字から右側全部の文字数が〈式 2〉より小さい場合は、〈文字列〉の〈式 1〉番目の文字より右側全部の文字列を抜き出します。

〈式 1〉が〈文字列〉の文字数より大きければ、MID\$の値はヌルストリング(空の文字列)となります。

注意: この命令は 2 バイト系日本語文字列にも利用できますが、その際には文字列の前後に入る KI/KO コードに注意してください。

また、1 バイト系と 2 バイト系の混在文字列に使用する場合は、文字の境界に充分注意しないと、予期しない結果となります。

参照: LEFT\$, MID\$, RIGHT\$, サンプルプログラム 23, 25

MKI\$ / MKS\$ / MKD\$

関 数

機 能 数値データをその数値の内部表現に対応した文字列に変換します。

書 式 MKI\$(〈整数値〉)
MKS\$(〈単精度実数値〉)
MKD\$(〈倍精度実数値〉)

文 例 A\$=MKI\$(16961)
LSET B\$=MKS\$(1.23)
RSET C\$=MKD\$(3.141592654 #)

これらの関数は、数値データをランダムデータファイルに対して書き出す際、数値データはそのままの形では書き出すことができないため、これを文字型データに変換するために使用するものです。

MKI\$: 整数値を 2 文字 (2 バイト) の文字列に変換 (CVI の逆)。

MKS\$: 単精度実数値を 4 文字 (4 バイト) の文字列に変換 (CVS の逆)。

MKD\$: 倍精度実数値を 8 文字(8 バイト)の文字列に変換(CVD の逆)。

実際にランダムデータファイルに数値データを書き出す際には、まずこれらの関数を用いて数値データを文字型化し、これを LSET/RSET でフィールド変数にセットしてから、PUT を行います。

これらの関数で文字列に変換された数値データを、もとの数値型に変換するには、CVI/ CVS/CVD の各関数を使用します。

数値から文字への変換は数値の内部表現の値をそのままそれに対応する文字コードをもつ文字列にすることによって行われます。実際の数値と変換後の文字列との関係を整数値で説明すると次のようになります。

整数値が 16961 の場合、この数値の内部表現は&H4241(16 進表記)であり、A\$=CHR\$(&H41)+CHR\$(&H42)、すなわち、A\$="AB"となります(上下バイトが逆になることに注意)。つまり、数値データをキャラクタコードとみなすわけです。

注意：STR\$関数とは変換のしかたがまったく異なります。

参照：CVI/ CVS/ CVD, LSET/ RSET, サンプルプログラム 26

MON (DISK モード)

機 能	モニタモードに入ります。
書 式	MON
文 例	MON

BASIC モードからモニタモードに移ります。モニタモードは、機械語プログラムの作成、デバッグのための環境を用意します。詳しくは BASIC ユーザーズマニュアルを参照してください。CTRL+B で、もとの BASIC モードにもどります。

MOTOR

機 能	カセットテープレコーダのモータの ON, OFF を制御します。
書 式	MOTOR [〈スイッチ〉]
文 例	MOTOR 1

〈スイッチ〉を 0 にすればモータは OFF となり、0 以外の値にすれば ON となります。また、〈スイッチ〉を指定しない場合には、モータがそのとき ON の状態ならば OFF に、OFF の状態ならば ON にします。

NAME

NAME (DISK モード)

機 能	ディスクファイルの名前を変更します。
書 式	NAME <旧ファイルディスクリプタ> AS <新ファイルディスクリプタ>
文 例	NAME "2 : SAMPLE . BAS" AS "2 : SAMPLE1"

<旧ファイルディスクリプタ>で表されているディスク上のファイルの名前を<新ファイルディスクリプタ>に変更します。

ディスクファイルに対するファイルディスクリプタの指定形式は,"[ドライブ番号:] ファイル名"です。

ドライブ番号が省略された場合、ドライブ1のディスクが処理の対象になります。

<旧ファイルディスクリプタ>と<新ファイルディスクリプタ>に指定されるドライブ番号は同じでなければなりません。

なお、ファイルの名前の変更を行うファイルはクローズされた状態でなければなりません。

NEW

書 式	メモリ上にあるプログラムを抹消し、すべての変数を初期化します。
書 式	NEW
文 例	NEW

NEW はメモリ上のプログラムを抹消し、すべての変数を初期化(クリア)します。また、そのとき開かれているファイルもすべて閉じられます。

コマンドの実行が終ると、いつもコマンドレベルにもどります。

NEW ON

機 能	システムを再起動します。
書 式	NEW ON <式>
文 例	NEW ON 1

<式>に指定された値により、いろいろな状態でシステムを再起動させます。

この命令は、ディップスイッチ SW2 の持つ機能に対応して意味づけられており、ディップスイッチの操作の一部を BASIC コマンドとして実行できるようにしたものです。

NEW ON では、ディップスイッチ SW2 のうち、2 番～4 番スイッチまでを制御することが

できます。

NEW ON のスイッチは図のようにそれぞれを 1 ビット (0 または 1) で表します。

ディップスイッチ SW 2 の機能

2 <システムモード>	0 : BASIC モード	1 : ターミナルモード
3 <桁数>	1 : 1 行 80 桁	0 : 1 行 40 桁
4 <行数>	1 : 1 画面 25 行	0 : 1 画面 20 行

<式> に指定する値は、以下の式で求められます。

$$\text{<式>} = 4 \text{ 番スイッチの値} * 2^3 + 3 \text{ 番スイッチの値} * 2^2 + 2 \text{ 番スイッチの値} * 2$$

たとえば、25 行×40 桁表示の状態では、N₈₈-BASIC を BASIC モードで起動させたいときには、<式>=1 * 2³+0 * 2²+0 * 2 となり、<式> には、8 を指定します。

OCT\$

関 数

機 能	10 進数を 8 進数に変換し、その文字列を得ます。
書 式	OCT\$(<数式>)
文 例	PRINT OCT\$(320)

<数式> の値を 8 進数に変換して、その文字列を得る関数です。<数式> の値は、-32768 から 32767 (または 0 から 65535) までの範囲になければなりません。<数式> の値と得られる文字列との対応は次のとおりです。

<数式> の値	得られる 8 進表記文字列
0~32767	: 0~ 77777
-32768~-1	: 100000~177777
32768~65535	: 100000~177777

参照 : HEX\$, VAL, サンプルプログラム 24

ON COM GOSUB

機 能 RS-232C 回線からの割り込みが発生したとき、分岐する処理ルーチンの開始行を指定します。

書 式 ON COM [(〈回線番号〉)] GOSUB 〈行番号〉

文 例 ON COM GOSUB 1000
ON COM(2) GOSUB *RECV

指定した RS-232C 回線に入力割り込みがあったときに分岐する処理ルーチンの開始行番号を定義します。

〈回線番号〉に指定できる値と意味は次のとおりです。(〈回線番号〉)を省略(カッコも含む)した場合は、第 1 回線となります。

- 1 : RS-232C 第 1 回線
- 2 : RS-232C 第 2 回線
- 3 : RS-232C 第 3 回線

ただし、2、3 は専用のインタフェースボードが必要です。

〈行番号〉には分岐させる開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンと同じで、RETURN で行います。たんに RETURN としたときは中断した所から再開し、後ろに行番号を指定したときはその行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、割り込みが発生したとき、

COM ON の状態でなければなりません。

ON COM GOSUB は、RS-232C 回線を OPEN した後実行されなければ有効となりません。

参照：COM ON/OFF/STOP, OPEN, RETURN

ON ERROR GOTO

機能 エラーが起こったときに分岐する処理ルーチンの開始行を定義します。

書式 ON ERROR GOTO <行番号>

文例 ON ERROR GOTO 1000

<行番号>には分岐させる処理ルーチンの開始行番号を指定します。

この命令を実行しておくと、エラーが起こったときに、指定した開始行から始まるエラー処理ルーチンに処理が移ります。エラー発生時のプログラム行番号は ERL に、エラーコードは ERR にセットされますので、エラー処理ルーチン内では、これらを利用して独自のエラー処理を行うことができます。

処理ルーチンから復帰するには、一般のサブルーチンとは異なり、RESUME を使います。

プログラムの実行終了後は、必ず ON ERROR GOTO 0 を実行して、エラー割り込みを無効にしておくようにしてください。さもないと、エラーが発生するたびに(プログラムがメモリ上にある限り、ダイレクトモードであっても)エラー処理ルーチンに処理が移ってしまいます。

ON ERROR GOTO 0 の実行後は、エラーが生じると通常どおりエラーメッセージが表示され、プログラムの実行は停止します。

エラー処理ルーチン内に ON ERROR GOTO 0 がある場合は、エラーによる分岐(エラートラップ)の原因となったエラーのエラーメッセージを表示し、プログラムの実行を停止します。

注意: <行番号>に指定された行が存在しない場合、“Undefined line number”エラーが起こります。

エラー処理ルーチンの中ではエラー割り込みは起こりません。エラー処理ルーチンの実行中にエラーが起きた場合には、それに対するエラーメッセージが表示され、実行は停止します。

参照: ERL/ERR, ERROR, RESUME, サンプルプログラム 22

ON...GOSUB / ON...GOTO

機能 指定されたいずれかの行に分岐します。

書式 1) ON <式> GOSUB <行番号> [, <行番号> ...]

2) ON <式> GOTO <行番号> [, <行番号> ...]

文例 ON A GOSUB 100, 200, 300, 400

ON N GOTO *ENTRY1, *ENTRY2, *ENTRY3, *ENTRY4

〈式〉の値に応じて、“〈式〉の値”番目の〈行番号〉に処理が移ります。たとえば、〈式〉の値が3であったなら、行番号の並びの3番目の行が分岐先となります。

ON…GOSUBの場合、〈行番号〉はサブルーチン(RETURNで終わっている必要がある)の開始行でなくてはなりません。

注意：〈式〉の値が負の場合には、“Illegal function call”エラーが起きますが、値が0あるいは並びの行番号より大きい場合には次の行に実行が移り、エラーは起きません。

参照：サンプルプログラム 29, 30

ON HELP GOSUB

機能

[HELP]キーによる割り込み処理ルーチンの開始行を定義します。

書式

ON HELP GOSUB 〈行番号〉

文例

ON HELP GOSUB 1000

この命令を実行しておくと、以後プログラムの実行中に[HELP]キーが押されるたびに、指定した開始行から始まる割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象として[HELP]キーを使っていることを除けば、他の割り込み定義命令(KEY, COM, PEN, STOPなど)と同様な使い方で利用できます。

〈行番号〉には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURNによって行います。たんにRETURNとしたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きのRETURNでは、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

INPUT実行中に割り込みが生じた場合は、RETURNだけで復帰させるとINPUTの次の命令から実行を再開しますから、行番号を指定するか、プログラムで適切な処理をしなければなりません。

本来この命令は、アプリケーションプログラムを作成する際、プログラム中にプログラムの使い方、入力の仕方などの説明を書いた表示ルーチンを用意しておき、ユーザーがそのプログラムの実行中、使い方がわからなくなったときに[HELP]キーを押して指示を受ける、といった用途のために用意されているものです。

注意：この命令によって割り込み処理ルーチンに分岐させるには、割り込みがあったとき
HELP ON の状態でなければなりません。

参照：HELP ON/OFF/STOP, RETURN

ON KEY GOSUB

機能	ファンクションキーによる割り込みルーチンの開始行を定義します。
書式	ON KEY GOSUB <行番号> [, <行番号> ...]
文例	ON KEY GOSUB 100, 200, 300, 400

この命令を実行しておくと、以後プログラムの実行中にファンクションキーが押されるたびに、各ファンクションキーに対応する指定開始行から始まる割り込みルーチンに処理が移ります。

<行番号>には、割り込みが発生したときに分岐させる処理ルーチンの開始行番号を、ファンクションキーの番号順に並べて指定します。<行番号>の並び中で、行番号の指定を省略した場合は、それに対応するファンクションキーによる割り込みは起こりません。したがって、たとえば ON KEY GOSUB 100,, 300 と指定した場合、**f.1** キーおよび **f.3** キー以外は割り込みの対象となりません。

ファンクションキーは 10 個ありますから、最大 10 個まで <行番号> を並べて書くことができます。行番号の代わりにラベル名を使うこともできます。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、割り込みが発生したとき、KEY ON の状態でなければなりません。

この命令を実行すると、ファンクションキー本来の機能(ファンクションキーを押すと KEY で定義した文字列が入力される機能)は無視されます。ただし、定義された文字列の内容は、CONSOLE で指定することにより表示することは可能です。

参照：KEY ON/OFF/STOP, RETURN, サンプルプログラム 30

ON PEN GOSUB

機 能 ライトペンが押されたときの割り込み処理ルーチンの開始行を定義します。

書 式 ON PEN GOSUB 〈行番号〉

文 例 ON PEN GOSUB *PENTEST

この命令を実行しておくと、以後プログラムの実行中にライトペンが押されるたびに、指定した開始行から始まる割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象としてライトペンを使っていることを除けば、他の割り込み定義命令(KEY, COM, HELP, STOP など)と同様な使い方で利用できます。

〈行番号〉には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって処理ルーチンに分岐させるには、PEN ON の状態でなければなりません。

参照：PEN ON/OFF/STOP, RETURN

ON STOP GOSUB

機 能 [STOP] キーによる割り込み処理ルーチンの開始行を定義します。

書 式 ON STOP GOSUB 〈行番号〉

文 例 ON STOP GOSUB 100

この命令を実行しておくと、以後プログラムの実行中に[STOP]キーが押されるたびに、指定した割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象として[STOP]キーを使っていることを除けば、他の割り込み定義命令(KEY, COM, HELP, PEN など)と同様な使い方で利用できます。

〈行番号〉には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行いま

す。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、STOP ON の状態でなければなりません。

この命令はアプリケーションプログラム実行中に、ユーザーが誤って **STOP** キーを押してしまい、プログラムの実行が中断されてしまうのを防止するためのものです。したがって、不用意にこの命令が実行されてしまうと、本来の **STOP** キーおよび **CTRL**+**C** の機能は失われ、プログラムの中断ができなくなってしまうます。アプリケーションプログラムの動作が完全になってから、この命令を使うようにしてください。

参照：RETURN, STOP/ON/OFF/STOP

ON TIME\$ GOSUB

機能 内蔵クロックによる割り込みの発生時刻と、そのとき分岐する処理ルーチンの開始行を定義します。

書式 ON TIME\$="時刻" GOSUB 行番号

文例 ON TIME\$="07:30:00" GOSUB *MORNINGCALL

この命令を実行しておくと、以後プログラムの実行中、指定時刻になったときに指定した開始行から始まる割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象として内蔵クロックを使っていることを除けば、他の割り込み定義命令(KEY, COM, HELP, STOP など)と同様な使い方で利用できます。

〈時刻〉には、割り込み時刻を、"hh:mm:ss"の形の文字列(時:分:秒を各2桁で表したものの、TIME\$の設定と同様)で設定します。なお、この命令で制御できる分解能は2秒ですから、±1秒単位の誤差が出ることがあります。

〈行番号〉には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプ

OPEN

ログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、TIMES\$ ON の状態でなければなりません。

なお、この命令は、実行された時点で、設定された時刻から現在の時刻を引算して割り込み時刻を決定します。したがって、この命令の実行後に TIMES\$ で現在時刻の変更を行うと、割り込み時刻に狂いが生じますので注意してください。

参照：RETURN, TIMES\$, TIMES\$ ON / OFF / STOP, サンプルプログラム 31

OPEN

機 能 ファイルを開きます。

書 式 OPEN <ファイルディスクリプタ> [FOR <モード>] AS [#]<ファイル番号>

文 例

```
OPEN "TEST.BAS" FOR INPUT AS #1
OPEN "LINES.DAT" FOR OUTPUT AS #2
OPEN "LINES.DAT" FOR APPEND AS #2
OPEN "2:DATA1" AS #1
OPEN "COM:N81XN" AS #1
```

この命令は、<ファイルディスクリプタ>で指定したファイルをオープンして入出力操作のための専用バッファを用意し、それに<ファイル番号>をつけます。以後、ファイルへの入出力は、<ファイル番号>を指示することにより行います。

プログラム中で、複数の OPEN を使うことにより、同時に複数のファイルをオープンすることができます。同時にオープンできるファイル数は、最高15ですが、BASIC の起動時に "How many files (0—15) ?" で指定した数を超えてオープンすることはできません。複数のファイルをオープンする場合、<ファイル番号>にはそれぞれ異なった数値を指定します。

<ファイル番号>には、1 から15までの数値を指定することができますが、BASIC 起動時に指定した数を超えてはなりません。

<モード>はファイルへのアクセス方法を指定するものです。モードには次の4種類があります。

INPUT : 入力モード。既存のファイルから入力を行うよう指示。
OUTPUT : 出力モード。新しくファイルを作り、出力を行うよう指示。

APPEND : 追加モード。既存のファイルの終わりに追加出力を行うよう指示。
省略時 : ランダムモード。ディスクファイルのランダム入出力、または RS-232C 回線ファイルへの入出力を行うよう指示。

FOR〈モード〉を指定した場合、シーケンシャルファイルに対して入出力を行うことを指示します。

FOR〈モード〉を省略した場合、ランダムファイルに対して入出力を行うことを指示します。

〈ファイルディスクリプタ〉に指定したファイルが RS-232C 回線ファイルの場合は、シーケンシャル入出力ですが、ランダム入出力と同様、FOR〈モード〉は省略してください。

ランダム入出力を行うためには、OPEN を実行して用意した専用バッファに対し、FIELD によりフィールド変数を割り当てなければなりません。

注意：INPUT(入力)モード、APPEND(追加)モードでは、指定されたファイルが存在しないと、OPEN 実行時に“File not found”エラーとなります。OUTPUT(出力)モードでは、常に指定された名前のファイルが新しく作られ、同一名のファイルがあった場合にはそのファイルは削除されて新しく作り直されます。ランダムモードでは、指定されたファイルが存在しないと新たに作られますが、すでに存在しているときは指定ファイルに対して入出力が行われます(OPEN 実行時にファイルの削除や破壊は起こりません)。

RS-232C 回線ファイルをオープンする場合は、〈ファイルディスクリプタ〉として、ファイル名 COM〈回線番号〉に続けて、パリティ、ワード長などを指定する必要があります。指定形式は次のとおりです。

COM [〈回線番号〉] : [〈パリティ〉] [〈ワード長〉] [〈ストップビット〉] [〈XON スイッチ〉] [〈S パラメータ〉]]]]]

〈回線番号〉は RS-232C 回線の番号を表し、1, 2, 3 で指定します。省略すると 1 となります。なお、2, 3 は、RS-232C 拡張インタフェースボードが必要です。

〈パリティ〉は E, O, N で表し、それぞれ偶数パリティ、奇数パリティ、パリティ無しを意味します。

〈ワード長〉は 1 文字を表すのに用いるビット数で、7 または 8 で指定します。7 を指定した場合には、必ず 〈パリティ〉で O または E を指定しなければなりません。また、8 を指定した場合 〈パリティ〉は N でなければなりません。

〈ストップビット〉はストップビット数を決めるもので、1, 2, 3 で指定します。1 は 1 ビット、2 は 1.5 ビット、3 は 2 ビットを意味します。

〈XON スイッチ〉は XON/XOFF による通信制御を行うことを指定します。スイッチとして X が指定された場合 XON/XOFF 制御を行います。N が指定された場合にはこの制御は行われません。

〈S パラメータ〉は〈ワード長〉に 7 を指定したときに、キャラクタコード 128 以上の文字(カナ文字など)の入出力ができなくなるのを補助するパラメータです。S または N で指定します。S を指定したとき入出力可能で、N を指定したときは入出力不可能になります。

OPEN で扱うことのできるファイル機器は次のとおりです。

ファイル	使用できるモード
ディスクファイル(ドライブ番号:)	INPUT, OUTPUT, APPEND, 省略
RS-232C コミュニケーションファイル (COM:)	省 略
キーボードファイル (KYBD:)	INPUT
スクリーンファイル (SCRN:)	OUTPUT
プリンタファイル (LPT:)	OUTPUT
カセットファイル (CAS:)	INPUT, OUTPUT

注意：スクリーンファイル(SCRN:)を指定した場合、2バイト系日本語文字を出力することはできません。

参照：CLOSE, サンプルプログラム 1, 26, 27, 28, 29

OPTION BASE

機 能	配列の添字の最小値を指定します。
-----	------------------

書 式	OPTION	BASE	0
			1

書 式 OPTION BASE 1

OPTION BASE は、配列の添字の最小値を 0 または 1 に指定します。この命令が用いられない場合、最小値は 0 となります。

この命令は、プログラム中で配列変数が宣言、または引用される前に置かなければ効果がありません。

また、添字の最小値を一度指定すると、再指定によって最小値を変更することはできません。
この指定を解除、あるいは変更するには、RUN あるいは CLEAR を行わなければなりません。

注意：OPTION BASE 1 を実行して添字の最小値を 1 に指定した場合、以後、配列の添字として 0 が用いられると、“Subscript out of range”エラーが起こるようになります。また、添字の最小値を再指定しようとすると、“Duplicate Definition”エラーになります。

参照：DIM, サンプルプログラム 5

OUT

機能 出力ポートに 1 バイトのデータを送出します。

書式 OUT <ポート番号>, <式>

文例 OUT 64, 32

<ポート番号> は出力ポートの番号で、0～32767(&H0～&H7FFF)の範囲内の整数で指定します。

<式> はポートに出力するデータで、0～255 の範囲内の整数で指定します。

ポート番号と装置との対応については、ハードウェアマニュアルを参照してください。

参照：INP

[1] PAINT

機能 指定された境界色で囲まれた領域を、指定された色でぬります。

書式 PAINT (Wx, Wy) [, <領域色> [, <境界色>]]
STEP(x, y)

文例 PAINT (-50, 120), 3, 4

PAINT STEP(10, 10), 6

(Wx, Wy)を中心点として、<境界色>で囲まれた領域を、指定された<領域色>でぬりつづけます。(Wx, Wy)はワールド座標で指定します。STEPをつけて、LP(最終参照点)からの相対座標(x, y)で指定することもできます。

<領域色>、<境界色>はともにパレット番号で指定します。<領域色>が省略された場合には、COLORで指定されたフォアグラウンドカラーが用いられます。<境界色>が省略された場合には、<領域色>の指定と同じパレット番号が<境界色>として用いられます。

(Wx, Wy)の色が境界色と同じ色であった場合には、PAINTは、何の画面操作も行いません。

注意：PAINTはビューポート内でのみ働きますので、ビューポートの境界はPAINT動作の境界とみなされます。また、(Wx, Wy)がウィンドウの外にある場合には、“Illegal function call”エラーとなります。

参照：[1] COLOR, VIEW, サンプルプログラム 17

[2] PAINT

機能 指定された境界色で囲まれた領域を、指定されたタイルパターンで埋めます。

書式 PAINT (Wx, Wy) , <タイルストリング> [, <境界色>]
STEP(x, y)

文例 PAINT (255, 120), TILE\$, 4

(Wx, Wy)を中心点として、<境界色>で囲まれた領域をタイルパターンで埋めます。(Wx, Wy)はワールド座標で指定します。STEPをつけて、LP(最終参照点)からの相対座標(x, y)で指定することもできます。

<タイルストリング>は、タILINGに用いられる基本タイルの大きさと模様を決定する文字列です。タイルの大きさは、横方向は8ドット分と決められていますが、縦方向の長さは<タイルストリング>の文字列の長さで決まります。縦方向がnドットのタイルを作るためには、<タイルストリング>として、白黒モードでn文字、8色中・8色カラーモードで3*n文字、4096色中・16色カラーモードで4*n文字の長さがそれぞれ必要です。

タイルの模様は、<タイルストリング>の文字列のキャラクタコードの2進数表現(ビットパターン)により表現されます。<タイルストリング>の指定の方法は白黒モードとカラーモードとで異なります。

●白黒モードの場合

<タイルストリング>中の各1文字(バイト)のコードがタイルの横8ドットの線に対応します。文字コードの2進数表現で、タイルの横8ドットのうち、1に対応するドットはセット(白)、0に対応するドットはリセット(黒)として表されます。<タイルストリング>がn文字の長さであれば、その文字列の表す模様は、このようにして決定される横8ドットのパターンを縦にn文字分だけ並べたパターンになります。

例)CHR\$(&HAA)+CHR\$(&H55)

16進数表現	2進数表現	ドットパターン
&HAA	10101010	●○○●○○●○
&H55	01010101	○●○●○●○●

2文字からなる簡単なタイルストリングの例です。この2文字は1であるビットと0であるビットが互いに逆になっています。このパターンを基本タイルとして指定された領域を埋めれば、細かい市松模様になりますが、スクリーンのドットが大変細かいため、灰色のように見えます。

●カラーモードの場合

白黒モードと同様に、タイルの模様は<タイルストリング>に対応するビットパターンによ

って決定されます。白黒モードと異なる点は、8色中・8色モードの場合は3文字分ごとに、また4096色中・16色モードの場合は4文字分ごとに、タイルの横8ドット(1ライン分)の色がパレット番号として決定されることです。色の決定のされ方については、次の例を参照してください。

例) 8色中・8色モードあるいは4096色中・8色の場合

CHR\$(&HAA)+CHR\$(&H55)+CHR\$(&HFF)

ドット	1	2	3	4	5	6	7	8	
&HAA	1	0	1	0	1	0	1	0	← 2 ⁰ の桁
&H55	0	1	0	1	0	1	0	1	← 2 ¹ の桁
&HFF	1	1	1	1	1	1	1	1	← 2 ² の桁
対応する パレット番号	5	6	5	6	5	6	5	6	

各ドットについて、縦方向に2進数としてパレット番号に読み換えます。この場合、指定するタイルストリングのうちの最初の文字が最下位ビット(2⁰)の桁を表し、以後、2¹の桁、2²の桁、となります。

この例では、パレット番号5とパレット番号6が交互に、横8ドットに指定されたことになります。したがって、パレットが初期状態であれば水色と黄色が交互に出る模様になります。

例) 4096色中・16色モードの場合

CHR\$(&HCE)+CHR\$(&H8C)+CHR\$(&HD0)+CHR\$(&H22)

ドット	1	2	3	4	5	6	7	8	
&HCE	1	1	0	0	1	1	1	0	← 2 ⁰ の桁
&H8C	1	0	0	0	1	1	0	0	← 2 ¹ の桁
&HD0	1	1	0	1	0	0	0	0	← 2 ² の桁
&H22	0	0	1	0	0	0	1	0	← 2 ³ の桁
対応する パレット番号	7	5	8	4	3	3	9	0	

前の例と同じように、各ドットについて、縦方向に2進数としてパレット番号に読み換えることにより、表のような結果となります。

注意：カラーモードの場合、〈タイルストリング〉の文字数が3(4096色中・16色モードの場合は4)の倍数でないときには、文字列の後ろから、3(あるいは4)で割った余りの数だけ無視されます。また、3文字(あるいは4文字)に満たない場合には“Illegal function call”

エラーとなります。

すでにタイリングされている領域を異なるパターンでタイリングしようとする、時間がかかったり、スタックを消費するために“Out of memory”エラーになったりすることがあります。

また、PAINT はビューポート内でのみ働きますので、ビューポートの境界が PAINT 動作の境界とみなされます。また、(Wx, Wy) がウィンドウの外にある場合には、“Illegal function call” エラーとなります。

PEEK

関 数

機 能 メモリ上の指定番地の内容を読み出します。

書 式 PEEK(<アドレス>)

文 例 A=PEEK(&H100)

指定されたメモリ上の番地から 1 バイト (8 ビット) のデータを読み出します。

<アドレス> は 2 バイトの値で、0～65535 (&H0～&HFFFF) の範囲で指定します。この値はこの命令実行前に DEF SEG で宣言されたセグメントベースからの相対アドレスを意味します。

参照：CLEAR, DEF SEG, POKE, VARPTR

PEN

関 数

機 能 ライトペンから情報を得ます。

書 式 PEN(<機能>)

文 例 X=PEN(1)

ライトペンが、現在どのような状態にあるかの情報を得ます。

ライトペンからの情報には、ライトペンが押され、光を感じた瞬間のもの (トリガセンス) とライトペンが押されて光を感じている間のもの (レベルセンス) があります。

<機能> に指定する値により、次のような情報が得られます。

0 : トリガセンス

ペンが押された瞬間、光を感じたら 1 (真)、光を感じないと 0 (偽) となる。この機能を指定した場合、PEN 関数は一度真になると ON PEN GOSUB で定義されている割り込み処理ルーチンに入るまでその値を保持し、処理ルーチンに入

ったときに偽となる。

1 : レベルセンス

ペンが押されて光を感じている限り、現在ペンが置かれているキャラクタ座標の水平座標を得る。

2 : レベルセンス

ペンが押されて光を感じている限り、現在ペンが置かれているキャラクタ座標の垂直座標を得る。

注意：〈機能〉に指定できる値は 0～2 の範囲で、それ以外は “Illegal function call” エラーとなります。またこの関数は PEN ON の状態でなければ値を得ることができません。

参照：ON PEN GOSUB, PEN ON/OFF/STOP

PEN ON/OFF/STOP

機能

ライトペンによる割り込みの許可、禁止、停止を制御します。

書式

1) PEN ON

2) PEN OFF

3) PEN STOP

文例

PEN ON

書式 1) 割り込みを許可します。以後ライトペンを押すごとに割り込みが起こり、ON PEN GOSUB で定義されている処理ルーチンに分岐します。

書式 2) 割り込みを禁止します。以後ライトペンを押しても処理ルーチンへの分岐は起こりません。

書式 3) 割り込みを停止します。以後ライトペンを押しても、そのことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかしその後 PEN ON によって割り込みが許可されると、前にライトペンを押したことが有効となり、処理ルーチンに分岐します。

注意：プログラム終了時には PEN OFF を実行するようにしてください。

参照：ON PEN GOSUB, PEN

POINT

機 能 LP(最終参照点)を変更します。

書 式 POINT (Wx, Wy) |
 STEP(x, y) |

文 例 POINT (−200, 30)
 POINT STEP(20, 20)

最後にグラフィック操作の行われた座標を「最終参照点」(Last Referenced Point, 略して LP)といいます。LP は相対座標により座標指定を行う際の基点となります。グラフィック関係の命令はほとんど、実行することにより、描画と同時にこの LP も変更します。

POINT はワールド座標(Wx, Wy)で LP をたんに設定,あるいは変更する働きをします。実際に図形を描いたり,図形に変化を与えることはありません。

STEP をつけると前の LP からの相対座標による指定となります。

例 1) LINE(−200, 30)−(100, 120), 3

例 2) POINT(−200, 30)
 LINE −STEP(300, 90), 3

例 1 と例 2 は,まったく同じ図形を描きます。

参照 : LINE, [1] POINT(関数), サンプルプログラム 17

[1] POINT

関 数

機 能 LP(最終参照点)の値を得ます。

書 式 POINT(<機能>)

文 例 WX=POINT(0)

最後にグラフィック操作の行われた座標(最終参照点, 略して LP)を, <機能> に指定する値により, スクリーン座標あるいはワールド座標として得ます。

<機能> には 0〜3 の値を指定します。

- 0 : ワールド座標系の値に変換された LP の X 座標
- 1 : ワールド座標系の値に変換された LP の Y 座標
- 2 : スクリーン座標系の値に変換された LP の X 座標
- 3 : スクリーン座標系の値に変換された LP の Y 座標

LP がウィンドウの外であった場合、POINT 関数によって得られる座標に対応する点は、画面上に表示されていない点であることに注意してください。またこのとき、得られた値がスクリーン座標系に変換された座標値であれば、ビューポート外になることに注意してください。

参照：POINT

[2] POINT

関 数

機 能 スクリーン座標系の指定された座標に表示されているドットの色を得ます。

書 式 POINT(Sx, Sy)

文 例 COL=POINT(473, 120)

スクリーン座標系の(Sx, Sy)で指定された座標に表示されているドットの色を、パレット番号で得ます。(Sx, Sy)がビューポートの外にある場合、[2]POINT の値は-1 となります。

白黒モードの場合、ドットの色はリセット(0：黒を表す)またはセット(1：白を表す)で表され、[2] POINT で得られる値はこのいずれかとなります。

POKE

機 能 メモリ上の指定番地へデータを書き込みます。

書 式 POKE <アドレス>, <式>

文 例 POKE &HF000, &HFF

指定されたメモリ上の番地に 1 バイト(8 ビット)のデータを書き込みます。

<アドレス>は 2 バイトの値で、0～65535(&H0～&HFFFF)の範囲で指定します。この値はこの命令実行前に DEF SEG で宣言されたセグメントベースからの相対アドレスを意味します。

<式>は書き込まれるデータで、0～255(&H0～&HFF)の値でなければなりません。もし小数点以下があると整数化(小数点以下を四捨五入)してから実行されます。

注意：この命令は、現在のメモリの内容を書き換えてしまうため、不用意に使うと BASIC が使っている作業領域を壊してしまい、誤動作の原因となることもあります。使うときには、メモリマップなどで使用可能な領域かどうかを確認してください。

参照：DEF SEG, PEEK, VARPTR

POS

関 数

機 能 テキスト画面上の現在のカーソルの桁位置を得ます。

書 式 POS(<数式>)

文 例 CX=POS(0)

現在のカーソルの桁(水平)位置をキャラクタ座標で得ます。値の範囲は 0～79(または 39)となります。得られる値は、0 が画面の左端の桁を、79(または 39)が右端の桁を、それぞれ意味します。

<数式>はダミーであり、どんな数式を指定しても結果に変わりはありませんが、省略することはありません。

参照：CSRLIN, サンプルプログラム 10

PRESET

機 能 画面上の任意の座標のドットを消去します。

書 式 PRESET | (Wx, Wy) | [, <パレット番号>]
| STEP(x, y) |

文 例 PRESET (100, 100)

ワールド座標(Wx, Wy)で示されるドットを消去します。

(Wx, Wy)の代わりに、STEP をつけて相対座標(x, y)で指定することもできます。

この命令は、オプションの<パレット番号>をつけない方が一般的な使い方です。その場合、(Wx, Wy)で表される座標のドットを、現在 [1] COLOR によって設定されているバックグラウンドカラーでぬり変えます(すなわちドットが消える)。

<パレット番号>を指定した場合は、PSET とまったく同じに機能し、そのパレット番号のカラーでドットを表示します。

PRESET を実行すると、LP(最終参照点)は、指定した座標点に変更されます。

参照：[1] COLOR, サンプルプログラム 17

PRINT

機 能 画面にデータを出力します。

書 式 PRINT [<式>] [, <式> ...] [,]
 ? ; ; ;

文 例 PRINT "COLUMN=" ; X, "LINE=" ; Y

<式>に、数値や文字列を指定すると、その数値、文字列がそのまま出力され、数値変数や文字変数が指定されると、それらの変数に代入されている数値や文字列が出力されます。<式>が省略された場合には改行のみを行います。

数値を出力する場合、その後ろに必ず空白が入ります。また、数値の前には符号用の桁が用意されていて、プラスのときには空白、マイナスのときにはマイナス符号が表示されます。

複数のデータを表示する場合、区切り記号として、コンマ(,), セミコロン(;), 空白を使います。データを表示する領域は、あらかじめ各行を 14 文字ごとに分割して定められており、区切り記号によって表示のしかたが変わります。

コンマ(,)を使った場合は、次の領域から表示し、セミコロン(;)あるいは空白を用いた場合には、直前に出力したもののすぐ後ろに出力します。

ダブルクォーテーション(")で囲まれた文字列の前後に、他の文字列や変数を並べる場合、その間の区切り記号を省略することができます。このときは、セミコロンと同様の働きをします。

<式>の最後にセミコロンやコンマを指定すると改行が起こらず、その行で次の PRINT による出力を続行します。

PRINT の短縮形として疑問符(?)を使うことができます。プログラム内で"? "を用いた場合、後で LIST を行くと、自動的に"PRINT" に変換されます。

注意：単精度の数値で、指数形式(浮動小数点形式)でなくても 6 桁以下の桁数で精度に影響をおよぼさず表示できるものは、通常的小数点形式(固定小数点形式)で表示されます。同様に倍精度の数値で 16 桁以下の桁数で精度に影響をおよぼさず表示できるものは、固定小数点形式の表示になります。また、表示する数値の長さが現在のカーソルの位置より後方にとれない場合(それを表示すると次の行にわたってしまう場合)は改行してからそれを表示します。

参照：LPRINT, PRINT USING, サンプルプログラム 12

PRINT

機能

ファイルにデータを書き出します。

書式

PRINT # <ファイル番号>, [<式>] [, | <式> ...] [, |
; ;]

文例

PRINT #2, A ; B ; C

PRINT #1, A\$; " , " ; B\$

出力モードでオープンしたファイル(シーケンシャルファイル、スクリーンファイル、プリンタなど)や、RS-232C 回線ファイルに、<式>により指定した文字列、数値などのデータを書き出します。<式>が省略された場合には、たんに改行コード(CHR\$(13))のみを書き出します。

<ファイル番号>には、OPEN によって、そのファイルをオープンしたときに使った番号を指定します。

<式> には、ファイルに書き込む数値式、文字式を指定します。

PRINT #は、PRINT で画面に出力する場合と同様の形式で、ファイルにデータを書き出しますが、とくにシーケンシャルファイルへの書き出しの際には、入力時にこれらのデータがファイルから正しく読み出せるような形式になるように、適切に区切らなければなりません(INPUT #参照)。

以下、ディスク上のシーケンシャルファイルに対して書き出しを行う場合につき、データの区切りかたについて説明します。

<式> が数値式の場合は、セミコロン(;)で区切ります。たとえば、

A=123

B=456

C=-78

PRINT #1, A ; B ; C

とすると、ディスクには次のように書き込まれます。

□ 123 □□ 456 □-78

(□は空白を意味します)

もし、区切り記号にコンマを使うとプリント領域の間に挿入される余分な空白もディスクに書き出してしまいます(挿入される空白の個数は PRINT の場合と同じです)ので、ディスクの記憶容量がむだ使いされます。

<式> が文字式の場合、セミコロンで区切り、かつ PRINT #中に独立した区切り記号を入れます(コンマをダブルクォーテーションで囲むなど)。たとえば、

```
A$="CAMERA"
B$="93604-1"
PRINT # 1, A$ ; ", " ; B$
```

とすると、ディスクには次のように書き出されます。

```
CAMERA, 93604-1
```

","を使わないで、PRINT # 1, A\$; B\$とした場合には、ディスクには CAMERA93604-1 と書き出されてしまいます。これでは区切りの記号がありませんから 2 つの別の文字列として読み込むことはできません。

文字式それ自身がデータとしてコンマ、セミコロン、意味のある始めの空白、キャリッジリターン(CHR\$(13)), またはラインフィード(CHR\$(10))などを含む場合は、ダブルクォーテーションコード(CHR\$(34))によって囲んでディスクに書き出さなければなりません。たとえば、

```
A$="CAMERA, □ AUTOMATIC"
B$="□□□ 93604-1"
PRINT # 1, A$ ; ", " ; B$
```

とするとディスクには、

```
CAMERA, □ AUTOMATIC, □□□ 93604-1
```

と書き出されます。これでは A\$の中のコンマがデータの区切りとなってしまう、INPUT # 1, A\$, B\$としてデータを読み込むと、A\$には CAMERA を、B\$には □ AUTOMATIC を読み込んでしまい、□□□ 93604-1 は読み込まれないで残ってしまいます。

これを避けるには、

```
PRINT # 1, CHR$(34) ; A$ ; CHR$(34) ; CHR$(34) ; B$ ; CHR$(34)
```

と指定します。この場合ディスクには、

```
"CAMERA, □ AUTOMATIC""□□□ 93604-1"
```

と書き出されますので、INPUT # 1, A\$, B\$とすれば、"CAMERA, □ AUTOMATIC"を A\$に、"□□□ 93604-1"を B\$に読み込むことができます。

注意：出力対象ファイルが "SCRN：" の場合、日本語データの出力はできません。

参照：INPUT #, PRINT # USING, WRITE #, サンプルプログラム 28

PRINT USING

機 能 文字列、数値などのデータを編集し、画面に出力します。

書 式 PRINT USING <書式制御文字列>; <式> [, | <式> ...] [, | ; |]

文 例 PRINT USING "####,."; A, B, C

<書式制御文字列>によって決定される領域や書式に従って、<式>に指定された各種データを画面に出力します。複数の<式>を並べる場合、区切り記号としてはコンマ(,)またはセミicolon(;)のいずれを使っても結果は変わりません。

<書式制御文字列>中に指定できる書式制御文字は、次のとおりです。

●文字変数や文字列の書式制御文字

!与えられた文字変数や文字定数の最初の1文字だけを出力します。

& <n 個の□(空白)> &...与えられた文字変数や文字定数の先頭から(n+2)文字の文字列を出力します。与えられた文字列が(n+2)文字より長い場合は余分な文字は無視され、短い場合には文字列は左づめで出力され、残った部分には空白が出力されます。

@.....文字列全体の出力に使います。複数個指定した場合、1つの"@ "に対して<式>の中の1つの文字列が、代入され出力されます。"@ "の数が<式>の個数より多い場合は、余った"@ "は無視されます。

注意: 日本語を含む文字列を編集することはできません。

●数値の書式制御文字

.....数値を出力する桁数を指定します。指定した桁数より数値の桁数が小さいときには、右づめで出力されます。

. (ピリオド) ...小数点の位置を指定します。小数点以下の部分で冗長となる桁には0が出力されます。

+ <書式制御文字列>の最初または最後につけた場合、数値の符号がそれぞれ前または後ろに出力されます。2個以上の"+ "を並べた場合には、余分は後述の制御文字以外の文字と同じ扱いとなり、そのまま出力されます。

- (マイナス記号) ... <書式制御文字列>の最後につけた場合、数値が負の数に数値の後ろに"- "が出力されます。前につけたり、2個以上並べた場合には後述の制御文字以外の文字と同じ扱いとなり、そのまま出力されます。

* * 〈書式制御文字列〉の先頭につけた場合、数値領域の左側に空白部分ができたとき、そこを“*”で埋めて出力します。この“**”は2桁分の領域を確保します。

¥ ¥ 〈書式制御文字列〉の先頭につけた場合、出力される数値の直前に“¥”を出力します。“¥¥”は2桁分の領域を確保しますが、このうち1桁分は“¥”の出力領域として使われます。後述の指数形式の書式指定を行った場合には、正しく出力されないことがあります。

* * ¥ 〈書式制御文字列〉の先頭につけた場合、上記の2つ(* *と¥ ¥)両方の機能となります。“**¥”は3桁分の領域を確保しますが、このうち1桁分は“¥”の出力領域として使われます。

, (コンマ) …桁数指定の“#”の並びの中においた場合、数値の整数部が3桁毎に“, ”で区切られて出力されます。ただし、上記の“. ”より右側においた場合は、数値の最後に“, ”が出力され、3桁毎の区切りは行われません。

^^^桁数指定の“#”の後につけた場合、指数形式(浮動小数点形式)で出力されます。

●その他の書式制御文字

_ (下線) …前述の制御文字1文字をたんに文字として表示するために使用します。“_”に続く1文字は常に書式制御機能を持たない文字として出力されます。

●制御文字以外の文字

前述の制御文字以外の文字を指定した場合、数値の前や後ろにそのキャラクタが出力されます。ただし、日本語文字は指定できません。

●数値の領域を超えた場合

指定した数値領域より数値の桁数が大きい場合、数値の直前に“%”が出力されます。数値を丸めた(四捨五入した)ことによって桁数が領域より大きくなった場合も、丸めた数値の前に“%”が出力されます。

●複数の書式制御文字を指定した場合

〈書式制御文字列〉中に複数の書式制御文字を指定した場合、その制御文字の並びが1つのパターンとなって、指定された複数の〈式〉に対し、順に繰り返してあてはめられます。

たとえば、

```
PRINT USING "@=¥¥### "; "BOOKS", 2500, "TICKETS", 1440, "DRINKS", 4300
```

とすると、“@=¥¥### ”のパターンが以降に指定される式の並びにあてはめられる結果、

PRINT # USING

BOOKS=¥2500 TICKETS=¥1440 DRINKS=¥4300

というように出力されます。

参照：LPRINT USING, PRINT, サンプルプログラム 13

PRINT # USING

機 能	文字列、数値などのデータを編集し、ファイルに出力します。
書 式	PRINT # <ファイル番号>, USING <書式制御文字列>; <式> [, <式> ...] [, ;]

文 例 PRINT #2, USING "@####";CASE\$, NUMBER

<ファイル番号>には、OPEN によって、そのファイルをオープンしたときに使った番号を指定します。

PRINT # USING は、<式> に指定した文字列や数値を <書式制御文字列> の形式にもとづいて編集し、ファイルに出力します。

この命令は、その対象がファイルであることを除けば PRINT USING と機能は同じです。

参照：LPRINT # USING, OPEN, PRINT #, PRINT USING

PSET

機 能	画面上の任意の座標にドットを表示します。
書 式	PSET (Wx, Wy) [, <パレット番号>] STEP (x, y)

文 例 PSET (100, 100), 4

ワールド座標(Wx, Wy)の位置にドットを表示します。
(Wx, Wy)の代わりに、STEP をつけて相対座標(x, y)で指定することもできます。
<パレット番号>でドットの色を指定します。省略した場合は、現在 [1] COLOR によって設定されているフォアグラウンドカラーが採用されます。

PSET を実行すると、LP(最終参照点)は、指定した座標点に変更されます。

参照：[1] COLOR, [2] COLOR, POINT, PRESET, サンプルプログラム 6, 17

PUT (DISK モード)

機 能 ファイルバッファ中のデータをファイルに書き出します。

書 式 PUT [#] <ファイル番号> [, <数式>]

文 例 PUT #3, COUNT

PUT 2

<ファイル番号> で指定されたファイルに、対応するバッファの内容を書き出します。

PUT は、指定されたファイルがディスクファイルか、またはプリンタあるいはスクリーンファイルかによってその動作が異なります。

(1) ディスクファイルの場合

PUT はランダムバッファ中のデータをランダムファイルに書き出します。指定されたファイルはランダムモードでオープンされていなければなりません。

<数式>はファイルのレコード番号として解釈され、指定されたレコードにバッファ中のデータが書き出されます。レコード番号の最小値は 1、最大値は 65000 です。<数式> が省略された場合には、直前に行われた、GET, PUT で指定されたレコードの次のレコードに書き出します。

(2) プリンタ(LPT : , LPT1 :)あるいはスクリーンファイル(SCRN :)の場合

バッファ中のデータを、プリンタあるいは画面に出力します。指定されたファイルは出力モードでオープンされていなければなりません。

<数式> は、バッファから、ファイルに対して書き出す文字(バイト)数と解釈されます。0 から 255 までの値で指定します。<数式> が省略されたとき、および 0 が指定されたときには 256 文字を書き出します。

注意：書き出すデータはあらかじめ FIELD, LSET/RSET により準備しておかねばなりません。

参照：FIELD, GET, LSET/RSET, OPEN, サンプルプログラム 1, 26, 29

PUT@

機能

グラフィックパターンや漢字を画面に表示します。

書式

- 1) PUT[@] (Sx, Sy), <配列変数名> [(<添字>)] [, <条件>] [, <フォアグラウンドカラー>, <バックグラウンドカラー>]
- 2) PUT[@] (Sx, Sy), KANJI(<漢字コード>) [, <条件>] [, <フォアグラウンドカラー>, <バックグラウンドカラー>]

文例

PUT@ (100, 100), G%, PSET

PUT (0, 0), KANJI(&H3021)

書式 1)

GET@によって配列変数に読み込まれたグラフィックパターンを、画面上の指定座標に表示します。

座標(Sx, Sy)はGET@の場合と同様で、ワールド座標でなくスクリーン座標で指定します。
<配列変数名>には、表示したいグラフィックパターンが格納されている配列変数の名前を指定します。

<添字>は、配列変数内に格納されている複数のグラフィックパターンのうち、どの要素から表示し始めるのかを指定するものです。省略した場合は、配列の最初から表示し始めます。<添字>の指定は、GET@でパターンを格納したときの値と対応させるようにしてください。

<条件>とは、グラフィックパターンを画面に表示する際のいろいろな条件を指定するものです。

PSET : 配列内のグラフィックパターンをそのまま表示します。

PRESET : 白黒モードの場合は配列内のパターンを反転して表示します。カラーモードの場合は各ドットのパレット番号を、7-(そのドットのパレット番号)(4096色中・16色モードの場合は、15-(そのドットのパレット番号))として表示します。

OR : 配列内のグラフィックパターンと、すでにある画面上のグラフィックパターンとをドットごとにOR(論理和)し、その結果を画面に表示します。

AND : 配列内のパターンと画面上のパターンをドットごとにAND(論理積)し、その結果を画面上に表示します。

XOR : 配列内のパターンと画面上のパターンをドットごとにXOR(排他的論理和)し、その結果を画面に表示します。

<条件>を省略した場合は、XORを指定したものとみなされます。

これらの<条件>は画面モードによって演算の対象が違います。白黒モードの場合、ドットがあるかないかを対象とし、カラーモードの場合、ドットごとのパレット番号を対象とします。たとえばすでに画面にあるドットの色がパレット番号3で、配列内のグラフィックパターンの色がパレット番号6のときに“AND”を指定すると、パレット番号2が表示されます(0011

AND 0110 → 0010).

〈フォアグラウンドカラー〉、〈バックグラウンドカラー〉は、白黒モードのときに GET@ で読み込んだパターンをカラーモードにおいて表示するときのみ、有効なオプションパラメータです。この2つのパラメータは両方とも指定するか、両方とも指定しないかのどちらかしか許されません。

〈フォアグラウンドカラー〉は、白黒モードで読み込んだ際に白であったドットに対しての色指定で、〈バックグラウンドカラー〉は、同様に黒であったドットに対しての色指定です。カラーモードにおいて、それぞれをパレット番号によって指定すると任意の色に変えることができます。

書式 2)

〈漢字コード〉で指定された漢字または非漢字をグラフィック画面に表示します。

使用できる文字は、JIS 第一水準、第二水準および利用者定義文字です。これらの中から任意の文字を〈漢字コード〉(JIS コード、ハードウェアマニュアル参照)によって指定することにより、画面上に日本語文字を表示することができます。使い方および機能は、〈配列変数名〉(〈添字〉)の代わりに KANJI(〈漢字コード〉)を使用することを除き、書式 1) の場合と同様です。

なお、利用者定義文字は、KPLOAD によってシステムに登録することができます。

書式 1)、2) とも、この命令を実行すると LP(最終参照点)は(Sx, Sy)に移動します。

注意: GET@ と PUT@ は、〈フォアグラウンドカラー〉、〈バックグラウンドカラー〉を指定して、白黒モードで読み込んだパターンをカラーモードで表示する用途の他は、原則として同一画面モードで使用するようになっています。

なお、1 つの PUT@ では 1 つの漢字しか表示することはできません。

参照: GET@, KPLOAD, サンプルプログラム 18, 19

RANDOMIZE

機能 新しい乱数系列を設定します。

書式 RANDOMIZE [〈式〉]

文例 RANDOMIZE 230

〈式〉に、新しい乱数の種(seed)を指定することによって、RND 関数で得られる乱数系列を変更します。〈式〉には数値や変数などを -32768 から 32767 の範囲で指定します。〈式〉を省略すると、メッセージが表示されて seed を要求してきますので、〈式〉に同様な範囲の値を入力

してください。もし、この範囲外の値を入力すると“Overflow”エラーとなります。

参照：RND，サンプルプログラム 32

READ

機 能	DATA で用意した数値や文字のデータを読み込み、変数に代入します。
書 式	READ <変数> [, <変数> ...]
文 例	READ ZIP, ADDR\$

READ は DATA と組み合わせて使わなければなりません。READ は DATA 行中に指定されたデータを 1 対 1 の対応で変数に割り当てていきます。

<変数>は、DATA 行中で指定したデータが文字定数である場合は文字変数でなくてはなりません。しかし、数値定数の場合は文字変数、数値変数のいずれにも読み込ませることができます。

1 つの READ で、複数の DATA 行を順番に参照したり、またいくつかの READ で 1 つの DATA 行を参照したりすることができます。

READ 中の<変数>の数が DATA のデータ数を超えてしまった場合は、“Out of DATA”エラーとなります。指定された<変数>が DATA のデータ数よりも少ない場合には、読まれなかったデータから、その次の READ が読み始めます。もしそれ以上 READ がない場合には、余分のデータは無視されます。

始めから、あるいは途中から DATA を読み直すには、RESTORE を使います。

注意：READ では、型が一致しない場合には、“Type mismatch”エラーではなく、“Syntax error”エラーが起こりますので注意してください。

参照：DATA，RESTORE，サンプルプログラム 3，14

REM

機 能	プログラムに注釈文を入れます。
書 式	REM [<注釈文>] ,
文 例	REM *** Main-Program ***

REM に続く文字列は、プログラムの注釈になるだけで、プログラムの実行にまったく影響を与えません。LIST を実行すると入力した内容がそのまま出力されます。REM の代わりにアポストロフィ(')を使うこともできます。

注意：REM に続く注釈文の後に、コロン(:)で区切って他の命令を続けても無視されてしまいます。

RENUM

機能 プログラムの行番号を新しくつけ直します。

書式 RENUM [<新行番号>] [, <旧行番号>] [, <増分>]

文例 RENUM 1000, 10

<新行番号> は、新しくつける行番号の最初の行番号で、省略すると 10 が採用されます。

<旧行番号> は行番号のつけ替えを始める現在のプログラムの行番号です。省略するとそのプログラムの最初の行番号が採用されます。

<増分> は新しくつける各行番号のあいだの増分で、省略すると 10 となります。

RENUM は、GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB および ERL で参照している行番号も新しい行番号に対応して変更します。もし、これらの命令の参照する行番号が存在しない場合には、次のようなエラーメッセージが表示されます。

```
"Undefined line xxxx in yyyy"
```

これは、行番号 yyyy 中に、参照されていない行番号 xxxx がある、という意味です。この場合、参照されていない行番号 xxxx は RENUM によって変更されませんが、行番号 yyyy は変更されてしまいますのでエラーの発生した行番号の所在が分からなくなってしまいます。したがって、RENUM を行う前にプログラムをセーブしておいたほうが安全です。

注意：65529 を超える行番号は発生することができません。このような場合には "Illegal function call" エラーとなります。

RESTORE

機能 READ で読む DATA 行の先頭行を指定します。

書式 RESTORE [<行番号>]

文例 RESTORE

RESTORE 800

〈行番号〉を指定すると、READ はその行番号の DATA 行からデータを読み始めます。省略した場合はプログラム中の最初の DATA 行から読み始めます。

参照：DATA，READ，サンプルプログラム 3

RESUME

機能 エラー処理ルーチンを終了し、元のプログラムの実行を再開します。

書式 1) RESUME [0]
2) RESUME NEXT
3) RESUME 〈行番号〉

文例 RESUME *START

書式 1) エラーの原因となった文からプログラムの実行を再開します。この場合 0 は省略できます。

書式 2) エラーの原因となった文の次の文からプログラムの実行を再開します。

書式 3) 〈行番号〉で指定した行から実行を再開します。

参照：ON ERROR GOTO，サンプルプログラム 22

RETURN

機能 サブルーチンを終了し、元のプログラムの実行を再開します。

書式 RETURN [〈行番号〉]

文例 RETURN
RETURN 200

サブルーチンの実行を終了し、そのサブルーチンを呼び出した GOSUB の次の命令から実行を再開します。1 つのサブルーチン内に複数の RETURN があってもかまいません。

RETURN では、〈行番号〉を指定して、特定の行にリターンさせることもできます。ただし、サブルーチンの多重化を行っている場合や、FOR～NEXT の内部から GOSUB でサブルーチンを呼び出している場合、リターン先の〈行番号〉には、そのサブルーチンの呼び出し側と同じスタックレベルのプログラム行を指定しなければなりません。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：RETURN を単独で使うことはできません。プログラムの実行中、GOSUB なしで RETURN に出会うと、“RETURN without GOSUB” エラーが起こります。

参照：GOSUB, サンプルプログラム 7

RIGHT\$

関 数

機 能 文字列の右側から任意の長さの文字列を抜き出します。

書 式 RIGHT\$(〈文字列〉, 〈式〉)

文 例 PRINT RIGHT\$("ABCDabcd", N*4)

〈文字列〉の右側(最後)から〈式〉で指定した桁数の文字列を抜き出します。〈式〉の値は 0 から 255 の範囲になければなりません。

〈式〉の値が 0 ならば、RIGHT\$ の値はヌルストリング(空の文字列)となります。〈式〉が〈文字列〉の文字数より大きければ、RIGHT\$ の値は〈文字列〉とまったく同じになります。

参照：LEFT\$, MID\$, サンプルプログラム 25

RND

関 数

機 能 乱数を得ます。

書 式 RND [(〈数式〉)]

文 例 R=RND

R=RND(-3)

0 以上 1 未満の乱数を得ます。得られる乱数は、RUN および CLEAR が実行されるごとにいつも同系列となります。ただし、RANDOMIZE を使用すれば、この系列を変えることが可能です。

RND 関数の機能は、指定する〈数式〉の値によって次のように異なります。

負の数 : 乱数系列を初期化する。

0 : 1 つ前に発生した乱数の値をとる(繰り返す)。

正の数 : 次の乱数を発生する。

なお、(〈数式〉)を省略して(カッコも含む)、たんに“RND”とした場合は、正の数を指定したのと同じ機能になります。

参照：RANDOMIZE, サンプルプログラム 32

ROLL

機 能 グラフィック画面を上下あるいは左右にスクロールさせます。

書 式 ROLL [<上下方向ドット数>] [, <左右方向ドット数>] [,

N
Y

]

文 例 ROLL -16, 32, Y
ROLL , 4

<上下方向ドット数>には、画面の縦方向のドットの数指定します。許される値の範囲は640×200ドットの画面モードでは-199から199、640×400ドットの画面モードでは-399から399までの値で、正の場合には上方向に、負の場合には下方向に、指定のドット数(絶対値)だけスクロールします。省略した場合は、縦方向のスクロールは行いません。

<左右方向ドット数>には、画面の横方向のドットの数指定します。許される値の範囲は-639から639までの値で、正の場合には左方向に、負の場合には右方向に、指定のドット数(絶対値)以下で最も大きい8の倍数と等しいドット数だけ、スクロールします。省略した場合は、左右のスクロールは行いません。

NまたはYを指定すると、スクロールにより新たに現れた領域を決められた色でクリアすることができます。Yを指定するとバックグラウンドカラーでクリアし、Nを指定するか省略したときには、パレット番号0でクリアします。

参照：サンプルプログラム 19

RUN

機 能 メモリにあるプログラムの実行を開始します。また、ディスクからプログラムをメモリにロードし、そのプログラムを実行します。

書 式 1) RUN [<行番号>]
2) RUN <ファイルディスクリプタ> [, R]

文 例 RUN *MAIN
RUN "2 : TEST"

RUNは、プログラムの実行に先立ち、変数をすべて初期化し、オープンされているすべてのデータファイルを閉じます。

書式 1) <行番号>を指定すると、メモリ上のプログラムの、その行から実行が始まります。省略すると、最も若い行番号の行から実行が始まります。プログラムの実行が終了とコマンドレベルにもどります。

書式 2) <ファイルディスクリプタ>で指定されたディスク上のプログラムをロードし、ただちに実行します。したがって、メモリ上にプログラムがあった場合、そのプログラムは消去されてしまいますので注意してください。プログラムの実行が終了とコマンドレベルにもどります。

R オプションをつけた場合には、プログラムの実行前にデータファイルは閉じられず、オープンされたままでプログラムが実行されます。

参照：CHAIN

SAVE

機能 メモリにある BASIC プログラムをファイルにセーブします。

書式 SAVE <ファイルディスクリプタ> [,

A
P

]

文例 SAVE "2 : MYPROG", A

<ファイルディスクリプタ>で指定されるファイル(ディスク, RS-232C 回線, カセット)に、メモリ上のプログラムをセーブ(書き出し)します。指定したファイルディスクリプタと同じ名前のファイルが存在した場合には、古い内容は失われ、新しいものに更新されます。

A オプションが指定された場合には、プログラムはアスキー形式でセーブされます。つまり、プログラムを LIST で表示したときと同じイメージでセーブが行われます。

A オプションの指定がない場合は、プログラムはバイナリ形式に圧縮されてセーブされます。つまり、メモリに格納されているときのイメージでそのままセーブされます。

P オプションが指定された場合には、プログラムは暗号化されたバイナリ形式でセーブされます。P オプションつきでセーブしたプログラムをメモリにロードして、LIST や EDIT により内容を見たり、変更しようとするとき、"Illegal function call" エラーとなります。

注意：アスキー形式のセーブは、バイナリ形式よりも、多くのファイルスペースを必要としますが、コマンドによってはプログラムがアスキー形式でセーブされていることが条件となることがあります。たとえば、MERGE コマンドは、アスキー形式のファイルを必要とします。また、アスキー形式でセーブされたファイルは、データファイルとして読み出すことができます。

P オプションによって一度プログラムが暗号化されると、これを解除する方法は用意されておらず、二度と内容の変更などはできなくなるので注意してください。

参照：LOAD, MERGE

SCREEN

機能	グラフィック画面に対して種々のモードを設定します。
書式	SCREEN [〈画面モード〉] [, 〈画面スイッチ〉] [, 〈アクティブページ〉] [, 〈ディスプレイページ〉]
文例	SCREEN 1, 0, 0, 7

〈画面モード〉は、カラー、白黒、分解能など、グラフィック画面の最も基本的なモードを設定するパラメータです。

指定値		分解能(横×縦)	使用可能ページ数
0	カラーモード	640×200	4
1	白黒モード	640×200	12(16*)
2	高分解能白黒モード	640×400	6(8*)
3	高分解能カラーモード	640×400	2

ここで、*のついた数値は、16色モードのときの値を表します。
〈画面スイッチ〉は、指定する値により、現在グラフィック画面に描かれているパターンなどを一時的に消去するものです。

- 指定値
- 0(または1) グラフィック画面の表示を行う。
 - 2(または3) 現在グラフィック画面に表示されている図形などを一時的に消去する。

〈アクティブページ〉と〈ディスプレイページ〉はグラフィック命令で書き込むページと表示するページの選択を行うためのものです。

〈アクティブページ〉と〈ディスプレイページ〉に指定できる値とその意味はパレットモードによって異なります(パレットモードに関しては [1] COLOR を参照してください)。

(1) 8色中・8色モードあるいは4096色中・8色モードの場合

〈アクティブページ〉にはグラフィック命令によって書き込むページを指定します。画面モードによって扱える画面数は異なります。

〈アクティブページ〉に指定できる値とページ番号との対応関係

指定値	画面モード	書き込まれるページ番号
0～3	カラーモード(0)	1～4
0～11	白黒モード(1)	1～12
0～5	高分解能白黒モード(2)	1～6
0, 1	高分解能カラーモード(3)	1, 2

〈ディスプレイページ〉には、画面モードに対応したページのうちのどのページを表示するかを指定します。

〈ディスプレイページ〉に指定できる値と表示されるページ番号との対応関係

指定値	カラーモード (0)	白黒モード (1)	高分解能白黒モード (2)	高分解能カラーモード (3)
0	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
1	ページ 1 のみ表示	ページ 1 のみ表示	ページ 1 のみ表示	ページ 1 のみ表示
2	ページ 2 のみ表示	ページ 2 のみ表示	ページ 2 のみ表示	×
3	×	ページ 1, 2 を合成表示	ページ 1, 2 を合成表示	×
4	×	ページ 3 のみ表示	ページ 3 のみ表示	×
5	×	ページ 1, 3 を合成表示	ページ 1, 3 を合成表示	×
6	×	ページ 2, 3 を合成表示	ページ 2, 3 を合成表示	×
7	×	ページ 1, 2, 3 を合成表示	ページ 1, 2, 3 を合成表示	×
8	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
9	×	ページ 4 のみ表示	×	×
10	×	ページ 4 のみ表示	×	×
11	×	ページ 4, 5 を合成表示	×	×
12	×	ページ 6 のみ表示	×	×
13	×	ページ 4, 6 を合成表示	×	×
14	×	ページ 5, 6 を合成表示	×	×
15	×	ページ 4, 5, 6 を合成表示	×	×
16	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
17	ページ 3 のみ表示	ページ 7 のみ表示	ページ 4 のみ表示	ページ 2 のみ表示
18	ページ 4 のみ表示	ページ 8 のみ表示	ページ 5 のみ表示	×
19	×	ページ 7, 8 を合成表示	ページ 4, 5 を合成表示	×
20	×	ページ 9 のみ表示	ページ 6 のみ表示	×
21	×	ページ 7, 9 を合成表示	ページ 4, 6 を合成表示	×
22	×	ページ 8, 9 を合成表示	ページ 5, 6 を合成表示	×
23	×	ページ 7, 8, 9 を合成表示	ページ 4, 5, 6 を合成表示	×
24	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
25	×	ページ 10 のみ表示	×	×
26	×	ページ 11 のみ表示	×	×
27	×	ページ 10, 11 を合成表示	×	×
28	×	ページ 12 のみ表示	×	×
29	×	ページ 10, 11 を合成表示	×	×
30	×	ページ 11, 12 を合成表示	×	×
31	×	ページ 10, 11, 12 を合成表示	×	×

×印：指定不可

(2) 4096 色中・16 色モードの場合

〈アクティブページ〉にはグラフィック命令によって書き込むページを指定します。画面モードによって扱える画面数は異なります。

〈アクティブページ〉に指定できる値とページ番号との対応関係

指定値	画面モード	書き込まれるページ番号
0～3	カラーモード (0)	1～4
0～15	白黒モード (1)	1～16
0～7	高分解能白黒モード (2)	1～8
0, 1	高分解能カラーモード (3)	1, 2

〈ディスプレイページ〉には、画面モードに対応したページのうちのどのページを表示するかを指定します。

〈ディスプレイページ〉に指定できる値と表示されるページ番号との対応関係

指定値	カラーモード (0)	白黒モード (1)	高分解能白黒モード (2)	高分解能カラーモード (3)
0	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
1	ページ 1 のみ表示	ページ 1 のみ表示	ページ 1 のみ表示	ページ 1 のみ表示
2	ページ 2 のみ表示	ページ 2 のみ表示	ページ 2 のみ表示	×
3	×	ページ 1, 2 を合成表示	ページ 1, 2 を合成表示	×
4	×	ページ 3 のみ表示	ページ 3 のみ表示	×
5	×	ページ 1, 3 を合成表示	ページ 1, 3 を合成表示	×
6	×	ページ 2, 3 を合成表示	ページ 2, 3 を合成表示	×
7	×	ページ 1, 2, 3 を合成表示	ページ 1, 2, 3 を合成表示	×
8	×	ページ 4 のみ表示	ページ 4 のみ表示	×
9	×	ページ 1, 4 を合成表示	ページ 1, 4 を合成表示	×
10	×	ページ 2, 4 を合成表示	ページ 2, 4 を合成表示	×
11	×	ページ 1, 2, 4 を合成表示	ページ 1, 2, 4 を合成表示	×
12	×	ページ 3, 4 を合成表示	ページ 3, 4 を合成表示	×
13	×	ページ 1, 3, 4 を合成表示	ページ 1, 3, 4 を合成表示	×
14	×	ページ 2, 3, 4 を合成表示	ページ 2, 3, 4 を合成表示	×
15	×	ページ 1, 2, 3, 4 を合成表示	ページ 1, 2, 3, 4 を合成表示	×
16	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
17	×	ページ 5 のみ表示	×	×
18	×	ページ 6 のみ表示	×	×
19	×	ページ 5, 6 を合成表示	×	×
20	×	ページ 7 のみ表示	×	×
21	×	ページ 5, 7 を合成表示	×	×
22	×	ページ 6, 7 を合成表示	×	×
23	×	ページ 5, 6, 7 を合成表示	×	×
24	×	ページ 8 のみ表示	×	×
25	×	ページ 5, 8 を合成表示	×	×
26	×	ページ 6, 8 を合成表示	×	×
27	×	ページ 5, 6, 8 を合成表示	×	×
28	×	ページ 7, 8 を合成表示	×	×
29	×	ページ 5, 7, 8 を合成表示	×	×
30	×	ページ 6, 7, 8 を合成表示	×	×
31	×	ページ 5, 6, 7, 8 を合成表示	×	×
32	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
33	ページ 3 のみ表示	ページ 9 のみ表示	ページ 5 のみ表示	ページ 2 のみ表示
34	ページ 4 のみ表示	ページ 10 のみ表示	ページ 6 のみ表示	×
35	×	ページ 9, 10 を合成表示	ページ 5, 6 を合成表示	×
36	×	ページ 11 のみ表示	ページ 7 のみ表示	×
37	×	ページ 9, 11 を合成表示	ページ 5, 7 を合成表示	×
38	×	ページ 10, 11 を合成表示	ページ 6, 7 を合成表示	×
39	×	ページ 9, 10, 11 を合成表示	ページ 5, 6, 7 を合成表示	×
40	×	ページ 12 のみ表示	ページ 8 のみ表示	×

指定値	カラーモード (0)	白黒モード (1)	高分解能白黒モード (2)	高分解能カラーモード (3)
41	×	ページ 9, 12 を合成表示	ページ 5, 8 を合成表示	×
42	×	ページ 10, 12 を合成表示	ページ 6, 8 を合成表示	×
43	×	ページ 9, 10, 12 を合成表示	ページ 5, 6, 8 を合成表示	×
44	×	ページ 11, 12 を合成表示	ページ 7, 8 を合成表示	×
45	×	ページ 9, 11, 12 を合成表示	ページ 5, 7, 8 を合成表示	×
46	×	ページ 10, 11, 12 を合成表示	ページ 6, 7, 8 を合成表示	×
47	×	ページ 9, 10, 11, 12 を合成表示	ページ 5, 6, 7, 8 を合成表示	×
48	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
49	×	ページ 13 のみ表示	×	×
50	×	ページ 14 のみ表示	×	×
51	×	ページ 13, 14 を合成表示	×	×
52	×	ページ 15 のみ表示	×	×
53	×	ページ 13, 15 を合成表示	×	×
54	×	ページ 14, 15 を合成表示	×	×
55	×	ページ 13, 14, 15 を合成表示	×	×
56	×	ページ 16 のみ表示	×	×
57	×	ページ 13, 16 を合成表示	×	×
58	×	ページ 14, 16 を合成表示	×	×
59	×	ページ 13, 14, 16 を合成表示	×	×
60	×	ページ 15, 16 を合成表示	×	×
61	×	ページ 13, 15, 16 を合成表示	×	×
62	×	ページ 14, 15, 16 を合成表示	×	×
63	×	ページ 13, 14, 15, 16 を合成表示	×	×

この〈アクティブページ〉と〈ディスプレイページ〉については、8色モードと16色モードでは、指定できる値とその意味が異なることに注意してください。

SCREEN を実行すると、そのとき設定されていたウィンドウ、ビューポート、LP(最終参照点)は初期状態にもどります。なお、LPの初期状態はワールド座標、スクリーン座標とも(0, 0)です。

参照：[1] COLOR, サンプルプログラム 20, 21

SEARCH(DISK モード)

関 数

機 能	配列変数の中から指定された値を捜し出し、その要素の順位を得ます。
書 式	SEARCH(〈配列変数名〉, 〈整数表記〉 [, 〈開始添字〉] [, 〈ステップ値〉])
文 例	NUM=SEARCH(A%, 100, 0, 3)

〈配列変数名〉で指定された配列変数の要素の中から〈整数表記〉で指定された値を捜し、最初に見つかった要素の添字を得ます。指定された値がその配列変数内で見つからなかった場合、SEARCHの値は-1となります。

〈配列変数名〉は整数型の一次元配列でなければなりません。またこの配列変数は SEARCHの実行に先だってDIMによって宣言されていなければなりません。

〈整数表記〉には捜したい値を指定します。この値は整数でなければならず、実数を指定した場合は整数化してから実行します。

〈開始添字〉には、配列中のどの要素から捜し始めるかを指定します。省略した場合には、配列の最初から捜し始めます。

〈ステップ値〉を指定した場合は、その間隔で捜します。省略した場合は1が採用されます。

SEARCH はランダムアクセスファイルのインデックスを捜し出したり、GET@、PUT@で用いる配列データを捜し出したりするのに使うことができます。

参照：DIM, OPTION BASE

SET (DISK モード)

機 能 ファイル属性のセット・リセットを行います。

書 式 1) SET <ドライブ番号>,

"P"
"R"

2) SET <ファイルディスクリプタ>,

"P"
"R"

3) SET # <ファイル番号>,

"P"
"R"

文 例 SET 1,"P"
SET "2:NOTWRITE","P"
SET #1,"R"

指定したドライブ、ファイル、ファイル番号に、"P"または "R"(属性文字)によって指定された属性をつけます。

"P" はライトプロテクト(書き込み禁止)の属性をつけます。この属性が指定されると、PRINT #, PUT, WRITE #などの書き込み動作が禁止されるとともに、ファイルの KILL もできなくなります。

"R" はリードアフターライト(書き込み確認)の属性をつけます。この属性が指定されると、書き込みを行った直後に読み出しを行い、正しく書き込みが行われたかの確認が行われるようになります。

これら以外の文字が属性文字として指定された場合には、そのとき、設定されている属性が解除されます。

書式 1) <ドライブ番号>が指定された場合には、指定されたドライブのディスク媒体に対して属性がつけられます。

書式 2) <ファイルディスクリプタ>が指定された場合には、指定されたファイルのみに属性がつけられます。同一ディスク中の他のファイルには影響をおよぼしません。

書式 3) <ファイル番号>が指定された場合には、そのファイルが開かれている間だけ、指定された属性が作用します。

参照：ATTR\$

SGN

関 数

機 能	符号を調べます。
書 式	SGN(<数式>)
文 例	PRINT SGN(RESULT)

<数式> に指定された式の値の正負により、SGNの値は次のようになります。

正の場合	：	1
0	：	0
負の場合	：	-1

参照：サンプルプログラム 9

SIN

関 数

機 能	正弦(サイン)を得ます。
書 式	SIN(<数式>)
文 例	X=RADIUS*SIN(ANGLE) PRINT SIN(-3.14159/2)

<数式> の値に対する正弦値を得ます。<数式> の単位はラジアンで指定します。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：ATN, COS, TAN, サンプルプログラム 6, 30

SPACE\$

関 数

機 能	任意の長さの空白文字列を得ます。
書 式	SPACE\$(〈数式〉)
文 例	QTTY\$=NUM\$+SPACE\$(10)+RES\$

〈数式〉に指定した数だけの空白(スペース、キャラクタコード&H20)が連なった文字列を得ます。〈数式〉の値は 0 から 255 までの範囲内であればなりません。

参照：SPC，サンプルプログラム 25

SPC

関 数

機 能	任意の数だけの空白を出力します。
書 式	SPC(〈数式〉)
文 例	PRINT "SUPER" ; SPC(10) ; "BASIC"

出力の対象となる行の中で、〈数式〉に指定した数だけの空白(スペース)を出力します。

ただし、SPC は、PRINT や LPRINT などの出力文中でのみ使用することができるもので、文字式としては使えません。

〈数式〉には、-32768 から 32767 までの範囲を指定できますが、負の数はすべて 0 とみなされます。また、正の数の場合でも、WIDTH で決められている現在の水平表示桁数以上の場合は、〈数式〉をその表示桁数で割った余りを値とします。

参照：SPACE\$, TAB

SQR

関 数

機 能	平方根を得ます。
書 式	SQR(〈数式〉)
文 例	DISTANCE=SQR(X*X+Y*Y)

〈数式〉に指定した値の平方根(スクエアルート)を得ます。

〈数式〉に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：サンプルプログラム 8

STOP

機能 プログラムの実行を一時中断し、コマンドモードにもどります。

書式 STOP

文例 STOP

STOP はプログラムの実行を一時中断するためのもので、プログラムのどこにおいてもかまいません。STOP を実行すると、次のメッセージが表示されます。

Break in 〈行番号〉

ここで、〈行番号〉は実行の中断されたプログラム行の行番号を表します。

STOP が実行されると、BASIC は常にコマンドレベルにもどります。また、CONT コマンドによりプログラムの実行を再開することができます。ただし、INPUT、LINE INPUT、INPUT\$関数などの入力文の途中でストップした場合、継続することはできません。

STOP は END とは異なり、そのときオープンされていたデータファイルをクローズしますので注意してください。

参照：CONT、END

STOP ON / OFF / STOP

機能 STOP キーおよび CTRL + C による割り込みの許可、禁止、停止を制御します。

書式

- 1) STOP ON
- 2) STOP OFF
- 3) STOP STOP

文例 STOP ON

書式 1) 割り込みを許可します。以後 STOP キーあるいは CTRL + C を押すごとに割り込みが発生し、ON STOP GOSUB によって設定された処理ルーチンに分岐します。

書式 2) 割り込みを禁止します。以後 STOP キーあるいは CTRL + C を押しても処理ルーチンへの分岐は起こりません (STOP キーあるいは CTRL + C は通常の動作になります)。

書式 3) 割り込みを停止します。以後 STOP キーあるいは CTRL + C を押しても押されたことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後、STOP ON によって割り込みが許可されると、前に STOP キーあるいは CTRL + C を押したことによって、処理ルーチンに分岐します。

参照：ON STOP GOSUB

STR\$

関 数

機 能	数値を文字列に変換します。
書 式	STR\$(〈数式〉)
文 例	LISTING\$=STR\$(NUM)+NAME\$

〈数式〉に指定した値を、文字列に変換します。

変換後の文字列の最初の文字は、〈数式〉の値が正ならば空白(スペース)となり、負ならば“-” (マイナス記号)となります。

〈数式〉には、整数型および実数型(倍精度、単精度)の数値を指定することができます。

注意：STRING\$と区別すること。

参照：STRING\$, VAL, サンプルプログラム 24

STRING\$

関 数

機 能	任意の文字を任意の数だけ連結した文字列を得ます。
書 式	STRING\$(〈式〉, 〈文字式〉) 〈数式〉)
文 例	A\$=STRING\$(50, "+")

〈文字式〉に指定した文字、または〈数式〉に指定したキャラクタコードに該当する文字を、〈式〉に指定した数だけ連結した文字列を得ます。

〈文字式〉の場合には最初の 1 文字のみが用いられます。〈数式〉の場合は、値が 0 から 255 の範囲内でなくてはなりません。

参照：STR\$

SWAP

機能 2 つの変数の値を入れ換えます。

書式 SWAP <変数>, <変数>

文例 SWAP A\$, B\$

どの型(整数型, 単精度型, 倍精度型, または文字型)の変数でも, SWAP によってその値を交換することができます。

ただし, 2 つの変数の型が一致していないと, "Type mismatch" エラーが起こります。

参照: サンプルプログラム 10

TAB

関数

機能 出力対象行の任意の位置まで空白を出力します。

書式 TAB(<数式>)

文例 PRINT "SUPER" ; TAB(10) ; "BASIC"

出力の対象となる行の行頭から, <数式>に指定した桁数分だけ, 空白(スペース)を空けます。

ただし, TAB は, PRINT や LPRINT などの出力文中でのみ使用することができるもので, 文字式としては使えません。

<数式>には, -32768 から 32767 までの範囲を指定できますが, 負の数はすべて 0 とみなされます。また, 正の数の場合でも, WIDTH で決められている現在の水平表示桁数以上の場合は, <数式> をその表示桁数で割った余りを値とします。

TAB では, SPC と違い, <数式>で指定する数は常に行頭からの桁数, つまり位置を表しますから, 作表などに使用すると便利です。

参照: SPC, サンプルプログラム 22, 30

TAN

関数

機能 正接(タンジェント)を得ます。

書式 TAN(<数式>)

文例 Y=X*TAN(ANGLE)

<数式> の値に対する正接値を得ます。<数式> の単位はラジアンで指定します。

〈数式〉に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：ATN, COS, SIN

TERM

機 能	システムをターミナルモードにします。
書 式	TERM "[COM:] [〈パリティ〉 [〈ワード長〉 [〈ストップビット〉 [〈XON スイッチ〉 [〈S パラメータ〉 [〈DEL コード処理〉 [〈RET キー処理〉 [〈受信 CR コード処理〉 [〈日本語シフトコード指定〉]]]]]]]" [, [〈モード〉] [, 〈変数領域の大きさ〉]]
文 例	TERM "COM:E71XS", F, 1000

BASIC モードからターミナルモードに制御を移行します。ターミナルモードでは、RS-232C 回線を介して他のコンピュータや機器との通信を行うことが可能となります。また、ターミナルモード時においても、リモート BASIC プロトコルにより、N₈₈-BASIC の機能を利用することができます。詳しくは BASIC ユーザーズマニュアルを参照してください。

COM: で、RS-232C 回線を使って外部機器と入出力を行うことを宣言します。なお、COM: は省略することができますが、省略した場合は、後に続くすべてのパラメータを指定しなければなりません。

各パラメータは、次のように設定します。

〈パリティ〉 は、通信時に使用されるパリティを決定します。

- E : 偶数パリティ
- O : 奇数パリティ
- N : パリティ無し

〈ワード長〉 は、1 文字を表すために必要なビット数です。

- 7 : 7 ビット (〈パリティ〉 で O または E を指定しなければなりません)
- 8 : 8 ビット (〈パリティ〉 で N を指定しなければなりません)

〈ストップビット〉 は、ストップビット数を決定します。

- 1 : 1 ビット
- 2 : 1.5 ビット
- 3 : 2 ビット

〈XON スイッチ〉は、XON/OFF による通信制御を行うことを指定します。

- X : XON/OFF 制御を行う。
- N : XON/OFF 制御を行わない。

〈S パラメータ〉は〈ビット〉に 7 を指定したときに、キャラクタコード 128 以上の文字(カナ文字など)の入出力を可能にするためのパラメータです。

- S : 入出力可能
- N : 入出力不可能

〈DEL コード処理〉は、受信した DEL コードの処理方法を決定するものです。

- B : BS コードに変換されて BS コードとして処理されます。
- N : NUL コードに変換されて NUL コードとして処理されます。

〈RET キー処理〉は、RETURN キーを押したときに送信するコードを決定します。

- C : CR コードを送信します。
- L : CR コード, LF コードをこの順に送信します。

〈受信 CR コード処理〉は、CR コードを受信したときの動作を決定します。

- C : CR コード受信時に復帰動作と改行動作を行う。
- L : CR コード受信時に復帰動作だけを行う。

〈日本語シフトコード指定〉は、英数かな文字列のなかで日本語文字列の始まりを表示するコード(KI コード)と、日本語文字の終わりを表示するコード(KO コード)とを決定します。

- P : KI コード=&H1B4B(ESC・K に相当), KO コード=&H1B48(ESC・H に相当)が使用される。
- I : KI コード=&H1A70(SB・p に相当), KO コード=&H1A71(SB・q に相当)が使用される。

〈モード〉は、通信を半二重で行うか、全二重で行うかを決定します。

- H : 半二重通信モード
- F : 全二重通信モード

〈変数領域の大きさ〉は、リモート BASIC プロトコルを利用する際に用いられる変数領域の大きさを決定します。省略された場合 1024 に設定されます。

ターミナルモードから BASIC にもどるときには、SHIFT キーを押しながら STOP キーを押します。

注意：〈変数領域の大きさ〉を除くすべてのパラメータは、省略された場合、BASIC を起動したときに設定されていたメモリスイッチ(ハードウェアマニュアル参照)の値が採用され

ます。また、途中のパラメータを省略して後のパラメータを指定する場合は、省略記号としてブランク(空白)を指定しなければなりません。

また、パラメータには、必ず大文字を使ってください。

ターミナルモードを RS-232C 第 2 回線および第 3 回線で使うことはできません。

TIME\$

関 数

機 能

時刻を得ます。

書 式

1) TIME\$

2) TIME\$="hh:mm:ss"

文 例

PRINT TIME\$

TIME\$="23:15:00"

TIME\$には常に現在の時刻が"hh:mm:ss"(時:分:秒)の形で入れられており、いつでもその内容を見ることができます。時刻表示は、24 時制です。

なお、書式 2)を用いることにより、時刻を変更することもできます。hh には 00~23, mm には 00~59, ss には 00~59 の範囲の整数文字列を指定します。

注意:時刻は、バッテリーバックアップによって自動的に更新され、正しく維持されるようになっています。不用意に時刻を変えないようにしてください。

参照: DATE\$, サンプルプログラム 31

TIME\$ ON/OFF/STOP

機 能

リアルタイムタイマによる割り込みの許可、禁止、停止を制御します。

書 式

1) TIME\$ ON

2) TIME\$ OFF

3) TIME\$ STOP

文 例

TIME\$ ON

書式 1) 割り込みを許可します。以後、設定された時刻になると割り込みが発生し、ON TIME\$ GOSUB によって設定された処理ルーチンに分岐します。

書式 2) 割り込みを禁止します。以後、設定された時刻になっても処理ルーチンへの分岐は起こりません。

書式 3) 割り込みを停止します。以後、設定された時刻になってもそのことを覚えているだけ

で、処理ルーチンへの分岐は起こりません。しかし `TIME$ ON` によって割り込みが許可されると、先ほどの割り込みで処理ルーチンに分岐します。

注意：プログラムの終了時には `TIME$ OFF` を実行しておいてください。

参照：ON `TIME$ GOSUB, TIME$, サンプルプログラム 31`

TRON/TROFF

機能 プログラムの実行状態を追跡します。

書式 1) TRON
2) TROFF

文例 TRON
TROFF

TRON を実行してからプログラムを RUN させると、以後実行しているプログラムの行番号がカギカッコ ([]) つきでテキスト画面に表示され続けます。

TRON 状態を中止するには、TROFF を実行します。なお、NEW を実行した場合も TRON 状態は解除されます。

USR

関数

機能 メモリ上に用意された機械語関数を呼び出します。

書式 USR[<番号>](<引数>)

文例 I=USR3(J)

呼び出したい機械語関数はあらかじめメモリ上に用意しておき、また DEF USR により、その実行開始アドレスを設定しておかななくてはなりません。機械語関数を用意するには、POKE や BLOAD を用いることができます。

<番号> には、DEF USR により定義された番号を 0 から 9 までの値で指定します。省略された場合には 0 と解釈されます。

<引数> には、BASIC から機械語関数に渡す変数、定数、式を指定します。

USR が実行されると、直前に実行された DEF SEG で指定されたセグメントベースに、DEF USR により定義された機械語関数の実行開始アドレス(相対アドレス値)が加えられた番地に実行が移されます。USR によって呼び出された機械語関数は、機械語の IRET 命令により BASIC に制御をもどすことができます。

なお、機械語プログラムを呼び出す方法としては、USR の他に CALL が用意されています。CALL では複数の引数を機械語プログラムに受け渡すことができます。

参照：BLOAD, CALL, CLEAR, DEF USR, POKE

VAL

関 数

機 能 文字列表記の数値を実際の数値に変換します。

書 式 VAL(<文字列>)

文 例 A=VAL("&H20")

<文字列> に指定した文字列表記の数値を、実際の数値に変換します。

<文字列> には、整数表記(8 進形式, 10 進形式, 16 進形式)および実数表記(通常的小数点形式, 指数形式)のいずれも指定できます。

<文字列>の最初の文字が+, -, &, または数字でなければ, VAL の値は 0 になります。また, <文字列>中に数字を表す文字以外の文字が含まれていると, その文字以降の文字は無視されます。ただし, 16 進表記ならば A~F も数字とみなされ, 8 進表記ならば 8 および 9 は数字とみなされないことに注意してください。

なお, 文字列中のスペースは無視されます。

参照：STR\$

VARPTR

関 数

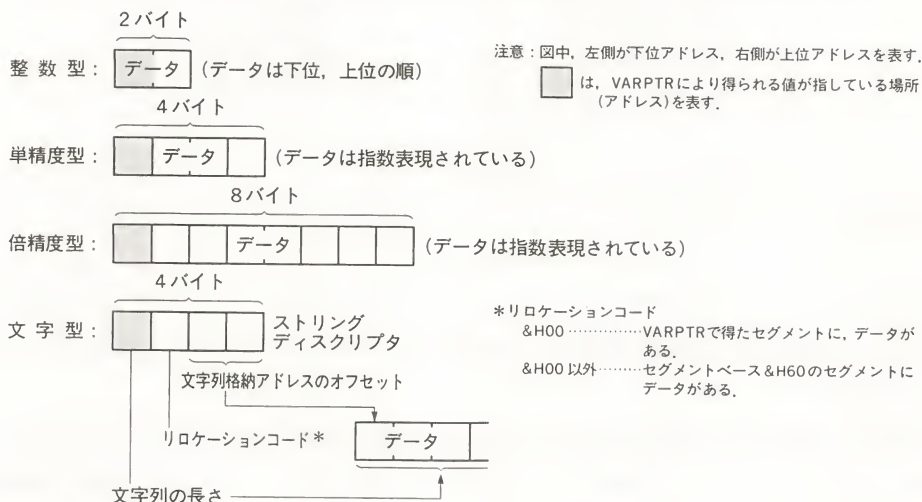
機 能 変数の値が格納されているメモリ番地, ファイルに割り当てられているファイルコントロールブロックの開始番地を得ます。

書 式 1) VARPTR(<変数名> [, <機能>])
2) VARPTR(# <ファイル番号> [, <機能>])

文 例 PRINT HEX\$(VARPTR(A, 1))
PRINT HEX\$(VARPTR(A))
SEGM%=VARPTR(# 1, 1)
ADDR%=VARPTR(# 1)

書式 1) <変数名>で指定した変数あるいは配列変数のデータが格納されている領域のメモリ番地を得ます。

<機能>に 0 を指定すると相対アドレスが得られ、0 以外の整数を指定するとセグメントベースが得られます。省略した場合は、0 が指定されたものとみなされます。



書式 2) 指定した<ファイル番号>に割り当てられているファイルコントロールブロックの開始番地を得ます。

<機能>に 0 を指定すると相対アドレスが得られ、1 を指定するとセグメントベースが得られます。省略した場合は、0 が指定されたものとみなします。

なお、ファイルに割り当てられている入出力バッファの開始番地は、このファイルコントロールブロックの 33 バイト目からの 2 バイトに格納されています。

参照: DEF SEG

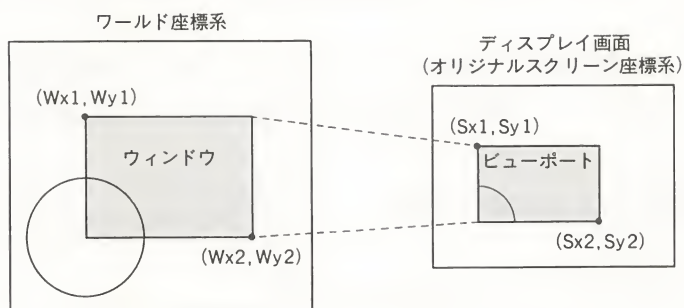
VIEW

機能	ディスプレイ画面上での表示領域(ビューポート)を指定します。
書式	VIEW (Sx1, Sy1)–(Sx2, Sy2) [, <領域色>] [, <境界色>]
文例	VIEW (100, 30)–(200, 75),, 7

オリジナルスクリーン座標系上の(Sx1, Sy1)を左上の頂点、(Sx2, Sy2)を右下の頂点とする長方形を図形表示領域として指定します。ここでいうオリジナルスクリーン座標とは、ディスプレイ画面のドットと 1 対 1 に対応して、WINDOW、VIEW により変化することのない物理的な座標のことです。

VIEW が実行されると、WINDOW により指定されているワールド座標系上の領域内の図形は、VIEW で指定した領域中に表示されるようになります。

この領域をビューポート (View Port) といいます。



〈領域色〉にパレット番号を指定すると、パレット番号の色でビューポート内をぬりつぶします。

〈境界色〉にパレット番号を指定すると、パレット番号の色でビューポートの枠を描きます。CLS によって消去される画面の範囲は、この枠によって囲まれた中だけで、枠は消されことはありません。

VIEW は、図形の表示領域の指定を行うだけで、実際の画面に対する操作は行いませんので、VIEW によりビューポートを変更することによって、以前のビューポート中の図形が移動するようなことはありません。また、VIEW はすべてのグラフィック画面表示の対象範囲を指定された領域に限定してしまいますから、ビューポートの外に点や線などを表示することはできません。

VIEW の指定を変えることにより、1つの図形を描くプログラムでも、画面上の異なる位置に異なる大きさで図形を描くことができます。

$Sx1 < Sx2$, $Sy1 < Sy2$ が成り立たない場合、あるいはこれらの座標がディスプレイ画面から外れている場合には、“Illegal function call” エラーとなります。

一度設定されたビューポートは、次に VIEW あるいは SCREEN が実行されるまで変化しません。また、VIEW は LP (最終参照点) をビューポートの左上の頂点に移動します。

注意：ワールド座標系がビューポート内に展開されるのは WINDOW の実行後であり、VIEW のみ実行してウィンドウは初期状態のままであれば、ビューポート内の座標は、(ワールド座標) = (スクリーン座標) となります。

参照：CLS, SCREEN, WINDOW, サンプルプログラム 20, 21

VIEW

関 数

機 能	現在のビューポートの設定位置を得ます。
書 式	VIEW(<機能>)
文 例	SX1=VIEW(0)

VIEW により設定されている現在のビューポートの設定位置(Sx1, Sy1 および Sx2, Sy2)を、〈機能〉に指定する値により、それぞれ個別に得ます。

〈機能〉には 0～3 の数値を指定します。

- 0 : ビューポートの左上の頂点の X 座標(Sx1)
- 1 : ビューポートの左上の頂点の Y 座標(Sy1)
- 2 : ビューポートの右下の頂点の X 座標(Sx2)
- 3 : ビューポートの右下の頂点の Y 座標(Sy2)

注意：VIEW 関数は、ディスプレイ画面上のビューポートの位置を得る関数ですから、得られる値は、スクリーン座標系の値となります。

参照：MAP, VIEW, WINDOW, WINDOW (関数)

WAIT

機 能	コンピュータの入力ポートをモニタする間、プログラムの実行を停止します。
書 式	WAIT <ポート番号>, <式 1> [, <式 2>]
文 例	WAIT 1, &H22, &HFF

WAIT は、指定した入力ポートのビットパターンが指定した状態になるまでプログラムの実行を停止します。

〈ポート番号〉は入力ポートの番号で、0～32767(&H0～&H7FFF)の範囲内で指定します。

WAIT は、まずポートから読み込んだデータと〈式 2〉との XOR をとり、次にその結果と〈式 1〉の AND をとります。もしその結果が 0(偽)なら、BASIC はもう一度ポートの状態を読み込み、同じ操作を繰り返します。もし結果が 0 でない(真)ならば、プログラムの実行は次の文に移ります。〈式 2〉を省略した場合は 0 とみなされます。

注意：WAIT の実行により、プログラムの実行が無限ループに入ってしまう場合があります。その場合にはコンピュータをリセットしなければなりません。

参照：INP, OUT

WHILE～WEND

機能

WHILE から WEND までの区間中にある一連の文を、指定条件が満足されている間、繰り返して実行します。

書式

```
WHILE <論理式>
  {
WEND
```

文例

```
WHILE J=<5
  {
WEND
```

WHILE～WEND ループ中(WHILE と WEND の間)におかれた文を、〈論理式〉が真である(0 でない)間、繰り返して実行します。〈論理式〉が偽(0)になると繰り返しは終わり、WEND に続く文に制御が移ります。

最初から〈論理式〉が偽の場合は、WHILE と WEND との間におかれた文は一度も実行されません。

WHILE～WEND は FOR～NEXT と同じ入れ子構造にすることができます。この場合には、それぞれの WEND はそれ以前の最も近くにある WHILE と対応します。WEND は WHILE と対になり、ループの終わりを示す働きをしますので省略することはできません。

注意：WHILE～WEND は必ず 1 対 1 に対応していなければなりません。

また、WHILE～WEND ループ内へ外部から GOTO などジャンプして入ってきたり、逆にループ内から外部へジャンプしたりするようなプログラムは、その動作が保証されなくなります。

参照：FOR…TO…STEP～NEXT, サンプルプログラム 11

WIDTH

機能

各種入出力機器やファイルに対して 1 行の長さなどを指定します。

書式

- 1) WIDTH <桁数> [, <行数>]
- 2) WIDTH <ファイルディスクリプタ>, <サイズ>
- 3) WIDTH # <ファイル番号>, <サイズ>

文例

```
WIDTH 80, 25
WIDTH "LPT1 :", 80
WIDTH # 1, 100
```

書式 1) テキスト画面に表示する文字の量を指定します。1 行あたり 40 文字、または 80 文字の〈桁数〉を設定します。また〈行数〉は 20 行か 25 行かのどちらかです。〈行数〉を省略した場合には、そのときの行数のままで変化しません。

書式 2) 〈ファイルディスクリプタ〉で指定したデバイスファイルに対して、1 行の長さを指定します。ここで指定可能なデバイスファイルは、RS-232C 回線ファイルとプリンタです。〈サイズ〉の値は 0 から 255 まで許されています。0 が 256 と解釈される以外は、指定した値が 1 行の文字数となります。たとえば、この命令がプリンタに対して実行されたとすれば、プリンタの 1 行に印字される文字数は〈サイズ〉で指定したものとなります。このようにプリンタに対して実行した場合、WIDTH LPRINT と同等の機能を得ることができます。初期状態では 255 に設定されています。

書式 3) 〈ファイル番号〉に割り当てられているバッファ (OPEN 参照) に対して、その大きさを〈サイズ〉で指定します。ここで指定できるデバイスファイルは、RS-232C 回線ファイルとプリンタです。〈サイズ〉は 0 から 255 まで許されています。0 が 256 と解釈される以外は、指定した値が 1 行の文字数となります。以後このファイル番号とデバイスとの入出力はこの〈サイズ〉単位で行われます。初期状態では 255 に設定されています。

なお、RS-232C 回線ファイルに対して WIDTH を実行すると、ファイルに対するデータの送出時、〈サイズ〉で指定された桁数の位置ごとに改行 (CR) コードを送出することになります。

注意：2 バイト系日本語 1 文字は文字数 2 文字 (バイト) として数えます。また、日本語文字列の前と後にはそれぞれ文字数で 2 文字分の制御コード (KI/KO) が挿入されますのでこれも計算に入れなければなりません。

参照：WIDTH LPRINT, サンプルプログラム 33

WIDTH LPRINT

機 能	プリンタに出力する 1 行あたりの文字数を設定します。
書 式	WIDTH LPRINT 〈文字数〉
文 例	WIDTH LPRINT 80

プリンタに印字する場合の、1 行あたりの文字数を設定します。〈文字数〉には 0 から 255 までの値を指定できます。0 が 256 と解釈される以外は、指定した値が 1 行の文字数となります。

注意：2 バイト系日本語 1 文字は文字数 2 文字 (バイト) として数えます。また、日本語文字列の前と後にはそれぞれ文字数で 2 文字分の制御コード (KI/KO) が挿入されますのでこれも計算に入れなければなりません。

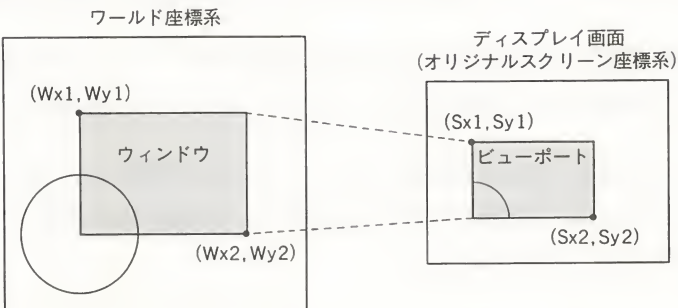
参照：WIDTH

WINDOW

機能	ビューポートに表示するワールド座標系内の領域を指定します。
書式	WINDOW (Wx1, Wy1) – (Wx2, Wy2)
文例	WINDOW (–300, –50) – (100, 70)

ワールド座標系上の(Wx1, Wy1)を左上の頂点、(Wx2, Wy2)を右下の頂点とする長方形で囲まれた領域を、ディスプレイ画面上のビューポート(VIEW で指定した領域)内に表示するよう設定します。

このようにして指定された領域はウィンドウと呼ばれ、ウィンドウから外れたところに描かれる図形は表示されなくなります。ウィンドウの大きさを変えると、対象領域の大きさが変わりますから、同じグラフィックス命令で描かれる図形でも、ビューポート内では異なった大きさで表示されることになります。



WINDOW も、VIEW と同じように領域の指定を行うだけで実際の画面に対する操作は行いませんので、WINDOW によってウィンドウの設定を変更するだけで、ビューポート中に表示される図形が変化するようなことはありません。また、図形が何も描かれない領域をウィンドウとして設定すると、ディスプレイ画面上のビューポートには何も描かれないことになりますので、注意してください。

$Wx1 < Wx2$, $Wy1 < Wy2$ が成り立たない場合、あるいはこれらの座標がワールド座標系から外れている場合には、“Illegal function call” エラとなります。WINDOW で指定する座標は、ワールド座標ですから、負の値や実数値もとることができます。

一度設定されたウィンドウは、次に WINDOW あるいは SCREEN が実行されるまで変化しません。また WINDOW は LP(最終参照点)をウィンドウの左上の頂点に移動します。

注意：ワールド座標系がビューポート内に展開されるのは WINDOW の実行後であり、VIEW のみ実行してウィンドウは初期状態のままであれば、ビューポート内の座標は、(ワール

ド座標)=(スクリーン座標)となります。

参照：VIEW

WINDOW

関 数

機 能 現在のウィンドウの設定位置を得ます。

書 式 WINDOW(<機能>)

文 例 WY1=WINDOW(1)

WINDOW により設定されている現在のウィンドウの位置(Wx1, Wy1 および Wx2, Wy2)を、<機能> に指定する値により、それぞれ個別に得ます。

<機能> には 0～3 の数値を指定します。

- 0 : ウィンドウの左上の頂点の X 座標(Wx1)
- 1 : ウィンドウの左上の頂点の Y 座標(Wy1)
- 2 : ウィンドウの右下の頂点の X 座標(Wx2)
- 3 : ウィンドウの右下の頂点の Y 座標(Wy2)

注意：WINDOW 関数で得られる値は、ワールド座標系の値となります。

参照：MAP, VIEW, VIEW(関数), WINDOW

WRITE

機 能 画面にデータを出力します。

書 式 WRITE <式> [, | <式> ...]
; |

文 例 WRITE LISTNUM, GOODS\$, COST

<式> に指定された数値式あるいは文字式の値を、画面に出力します。

<式>はコンマ(,)あるいはセミコロン(;)により区切ります。PRINT と違い、コンマとセミコロンとの機能上の区別はありません。

WRITE は、ほぼ、PRINT と同じように式の値を出力しますが、不要な空白桁は詰め、それぞれの式の値の間は必ずコンマで区切ります。また、文字列はダブルクォーテーション(")で囲んで出力します。

WRITE は、各式の値を出力したのち、改行を行います。

参照：PRINT, WRITE #, サンプルプログラム 12

WRITE

機 能 ファイルにデータを書き出します。

書 式 WRITE # <ファイル番号>, <式> [, | <式> ...]
; |

文 例 WRITE # 1, LISTNUM, GOODS\$, COST

出力モードでオープンしたファイル(シーケンシャルファイル, スクリーンファイル, プリンタなど)や, RS-232C 回線ファイルに, <式> により指定した文字列, 数値などのデータを書き出します。

<ファイル番号>には, OPEN によって, そのファイルをオープンしたときに使った番号を指定します。

<式>を複数指定する場合は, コンマ(,)あるいはセミコロン(;)により区切ります。PRINT #と違い, コンマとセミコロンとの機能上の区別はありません。

WRITE #は, ほぼ, PRINT #と同じように式の値を出力しますが, 不要な空白桁は詰め, それぞれの式の値の間は必ずコンマで区切ります(コンマそのものをデータとして出力します)。また, 文字列はダブルクォーテーション(")で囲んで出力します。

WRITE #は, 各式の値を出力したのち, 改行コード(CHR\$(13))を書き出します。

WRITE #は, 区切り記号としてコンマを必ず出力し, 不要な空白の出力は行わないため, PRINT #に比べファイルの使用領域を節約することができます。

注意：出力対象ファイルが"SCRN："の場合, 日本語データを出力することはできません。

参照：PRINT #, WRITE

サンプルプログラム

第4章 サンプルプログラム

この章の見方

この章には、N₈₈-BASIC(86)の各種命令・関数を使用したサンプルプログラムが掲載されています。

それぞれのプログラムは、相互に関連のある命令・関数を使って構成してあります。また、ポイントとなる各命令・関数には、右側に解説がつけてあります。

なお、各プログラムの番号は、「第3章 命令リファレンス」中の各命令の“参照”部分に示されたサンプルプログラム番号と対応しています。

以下に、本章中に掲載されているサンプルプログラムの番号・タイトルと、各プログラム中で使用されている主な命令の、一覧を示します。

- | | |
|--|--|
| 1. プログラムの連結 (その1)
CHAIN, COMMON, OPEN, FIELD,
PUT | 8. 条件分岐
IF...THEN~ELSE, ABS, SQR |
| 2. プログラムの連結 (その2)
CHAIN, COMMON, FIELD, GET | 9. 式の値による分岐
ON...GOTO, SGN |
| 3. DATA 行の読み込み
READ, DATA, RESTORE | 10. 変数の値の入れ換え
SWAP, LOCATE, POS, CSRLIN |
| 4. 変数の型宣言
DEFINT, DEFSNG, DEFDBL | 11. WHILE~WEND ループ
WHILE~WEND |
| 5. 配列変数の宣言と OPTION BASE
DIM, OPTION BASE, ERASE | 12. PRINT と WRITE
PRINT, WRITE |
| 6. FOR~NEXT ループ
FOR...TO...STEP~NEXT, SIN, COS,
PSET | 13. データの編集出力
PRINT USING |
| 7. サブルーチンの呼び出しとリターン
GOSUB, RETURN | 14. 円の描画
CIRCLE, READ, DATA |
| | 15. パレット (8色中・8色モード)
{1}COLOR, LINE, {2}COLOR |

16. パレット (4096 色中・16 色モード)

[1]COLOR, LINE, [2]COLOR

17. DRAW

POINT, DRAW, PSET, PRESET,
LINE, PAINT

18. グラフィックパターンの読み込みと表示

GET@, PUT@, LINE, CIRCLE

19. グラフィック画面のスクロール

PUT@, ROLL

20. ビューポートの設定

SCREEN, WINDOW, VIEW, LINE

21. ウィンドウの設定

SCREEN, WINDOW, VIEW, CIRCLE

22. エラー処理ルーチン

ON ERROR GOTO, ERROR, ERR,
RESUME

23. 文字とキャラクタコードの変換

ASC, MID\$, CHR\$

24. 8進変換と16進変換

OCT\$, HEX\$, STR\$

25. 文字列の抜き出し

LEFT\$, RIGHT\$, MID\$, LEN, SPACE\$

26. 数値データの文字型化

MKI\$, MKS\$, MKD\$, OPEN, CLOSE,
FIELD, LSET, PUT

27. 文字型化数値データの数値化

CVI, CVS, CVD, OPEN, CLOSE,
FIELD, GET

28. シーケンシャルファイル

OPEN, CLOSE, PRINT #, INPUT #,
EOF

29. ランダムファイル

OPEN, FIELD, CLOSE, LOF, GET,
LSET, PUT, INPUT\$, INKEY\$

30. キー割り込みルーチン

ON KEY GOSUB, KEY(n) ON,
RETURN, SIN, COS

31. タイマー割り込みルーチン

TIME\$, TIME\$ ON,
ON TIME\$ GOSUB, RETURN, BEEP

32. 乱数の発生

RANDOMIZE, RND

33. テキスト画面のモード設定と文字に対する色・機能の設定

CONSOLE, WIDTH, LOCATE,
COLOR@, GOSUB, RETURN

34. 利用者定義文字の登録

KPLOAD, KNJ\$

35. 2バイト系日本語文字列の変換と抜き出し

AKCNV\$, KACNV\$, KMID\$

36. 2バイト系日本語文字列と漢字コードの相互変換

JIS\$, KNJ\$, KMID\$

37. 特定タイプの文字列の抜き出し

KEXT\$

38. 2 バイト系日本語文字を含む文字列の長さ
とタイプ

KLEN, KTYPE

39. 利用者定義関数

DEF FN

1. プログラムの連結(その1)

```

100 DIM IDENT(200)
110 COMMON IDENT(),COUNT ————— 引き渡す変数を宣言。
120 OPEN "DATA" AS #1 ————— "DATA" ファイルをランダムモードでオープン。
130 FIELD #1,4 AS I$,20 AS N$,50 AS A$ — 会員番号, 氏名, 住所の各フィールド変数を定義。
140 INPUT "会員番号(1-999)";I
150 IF I=0 THEN GOTO *PRINTOUT
160 IF I>1000 THEN STOP
170 INPUT "氏名";NAMEI$
180 INPUT "住所";ADR$
190 LSET I$=MK$$(I)
200 LSET N$=NAMEI$
210 LSET A$=ADR$
220 PUT #1,I ————— 各データを各フィールド変数にセット。
230 COUNT=COUNT+1
240 IDENT(COUNT)=I ————— 会員番号と同番のレコードにデータを書き出す。
250 PRINT:GOTO 140
260 *PRINTOUT
270 CHAIN "CHAIN.S02" ————— 書き出された会員番号(レコード番号)を憶えておく。
280 END ————— "CHAIN.S02" に実行を移す。
                          "DATA" ファイルはオープンしたまま。

```

2. プログラムの連結(その2)

```

100 COMMON IDENT(),COUNT ————— 引き渡される変数を宣言。
110 PRINT:PRINT
120 FIELD #1,4 AS I$,20 AS N$,50 AS A$ — "DATA" ファイルはOPEN状態のままCHAIN
130 IF COUNT=0 THEN PRINT "END":END ————— されるので、新たにOPENしなくてよい。
140 GET #1,IDENT(COUNT) ————— 実際に書き出されているレコードのみを読み込む。
150 PRINT "会員番号:";CVS(I$)
160 PRINT "氏名:";N$
170 PRINT "住所:";A$
180 COUNT=COUNT+1
190 PRINT:GOTO 130

```

3. DATA 行の読み込み

```

100 READ A,B,C
110 RESTORE ————— 最初のDATA行(190行)から読み出すように指定。
120 READ D,E,F
130 RESTORE *STRDATA — *STRDATA行(210行)以降のDATA行(220行)から読み出すように指定。
140 READ A$,B$,C$
150 PRINT A,B,C
160 PRINT D,E,F
170 PRINT A$,B$,C$
180 END
190 DATA 1,2,3
200 DATA 4,5,6 ————— このプログラムでは、結局このDATA行は読み出されない。
210 *STRDATA
220 DATA AA,BB,CC

```

4. 変数の型宣言

```

100 DEFINT J-M ————— J~Mで始まる変数を整数型として定義。
110 DEFSNG A,B ————— A, Bで始まる変数を単精度実数型として定義。
120 DEFDBL D ————— Dで始まる変数を倍精度実数型として定義。
130 J1=1.23:K=65.643 ————— J1, K には整数に変換された値が代入される。
140 ABC=1.23:BBB=65.634
150 D=3.141592654000003#
160 D%=3.141592654000003# ————— D%は、整数型宣言文字(%)が付加されているため、
170 PRINT "J1=";J1,,"K=";K ————— 120行の宣言にかかわらず、整数型として扱われる。
180 PRINT "ABC=";ABC,,"BBB=";BBB
190 PRINT "D=";D,"D%=";D%
200 END

```


5. 配列変数の宣言と OPTION BASE

```

100 OPTION BASE 1 ————— 添字の最小値を1とする。
110 DIM KEISAN(9,9) ————— 要素数9×9の2次元配列変数KEISAN()を宣言。
120 FOR I=1 TO 9:FOR J=1 TO 9
130   KEISAN(I,J)=I*J
140 NEXT J,I
150 FOR I=1 TO 9:FOR J=1 TO 9
160   PRINT USING "####";KEISAN(J,I);
170 NEXT J:PRINT
180 NEXT I
190 PRINT:PRINT
200 ERASE KEISAN ————— KEISAN()を消去。
210 DIM KEISAN(9,9) ————— KEISAN()を新たに宣言。
220 FOR I=1 TO 9:FOR J=1 TO 9
230   KEISAN(I,J)=I+J
240 NEXT J,I
250 FOR I=1 TO 9:FOR J=1 TO 9
260   PRINT USING "####";KEISAN(J,I);
270 NEXT J:PRINT
280 NEXT I
290 END

```

6. FOR～NEXT ループ

```

100 SCREEN 0:CLS 3
110 FOR P=1 TO 7:C=0 —————
120   FOR R=10 TO 200 STEP 20 —————
130     FOR I=0 TO 3.14 STEP .05 —————
140       Y=SIN(I)*COS(I) ————— 楕円をプロットする。
150       X=SIN(I)*SIN(I) ————— 楕円の大きさを変える。
160       Y=Y*R+100 —————
170       X=X*R*2.5+100-C —————
180       PSET (X,Y),P —————
190     NEXT I —————
200     C=C+10 ————— 描き始める位置を変える。
210   NEXT R,P —————
220 END

```

バレット番号を変える。

7. サブルーチンの呼び出しとリターン

```

100 *START
110 INPUT "底辺:";TEIHEN
120 INPUT "高さ:";TAKASA
130 GOSUB *MENSEKI
140 PRINT "面積は ";MENSEKI
150 PRINT
160 GOTO *START
170 *MENSEKI ————— 三角形の面積を計算するサブルーチン。
180 MENSEKI=TEIHEN*TAKASA/2
190 RETURN ————— サブルーチンの最後はRETURNで終わる。

```

8. 条件分岐

```

100 INPUT A
110 FLG=0
120 IF A<0 THEN A=ABS(A):FLG=1 ————— 入力された値が負ならばFLGに1を代入。
130 PRINT SQR(A); ————— 平方根の計算。
140 IF FLG THEN PRINT "i" ELSE PRINT ————— 入力値が負のときには、平方根の結果に
150 END ————— "i" (複素表示)を付加する。

```

9. 式の値による分岐

```

100 *ENTRY
110 INPUT "N=";N
120 SG=SGN(N)+1 —— 入力値に対するSGN関数の値-1, 0, +1を, 0, 1, 2に変換.
130 ON SG GOTO *ZERO,*PLUS —— 入力値の負, 0, 正に従って, *MINUS,
140 *MINUS:PRINT "MINUS!":GOTO *ENTRY *ZERO, *PLUSに飛ぶ.
150 *ZERO:PRINT "ZERO!":GOTO *ENTRY
160 *PLUS:PRINT "PLUS!":GOTO *ENTRY

```

10. 変数の値の入れ換え

```

100 DIM A(10),B(10)
110 CLS
120 FOR I=1 TO 10
130 A(I)=I*2:B(I)=I*3:GOSUB *PRINTAB
140 NEXT I:LOCATE 0,Y+3
150 FOR I=1 TO 10
160 SWAP A(I),B(I):GOSUB *PRINTAB —— A()とB()の各要素をすべて交換.
170 NEXT I:LOCATE 0,Y+2
180 END
190 *PRINTAB —— A()とB()をそれぞれ表示するサブルーチン.
200 X=POS(0):Y=CSRLIN
210 LOCATE X,Y:PRINT USING "###";A(I);
220 LOCATE X,Y+1:PRINT USING "###";B(I);
230 LOCATE X+5,Y
240 RETURN

```

11. WHILE～WEND ループ

```

100 I=1
110 WHILE I<=20 —— Iが20以下のうちは120行～190行を繰り返す.
120 PRINT USING "## -";I;
130 J=1
140 WHILE J<=I —— Jが以下のうちは150行～170行を繰り返す.
150 PRINT USING "###";J;
160 J=J+1
170 WEND
180 PRINT:I=I+1
190 WEND

```

12. PRINT と WRITE

```

100 A%=123
110 B!=9.87654E+31
120 C#=3.14159265359#
130 D$="N88-BASIC"
140 PRINT A%;B!,C#;D$ —— PRINTとWRITEでは出力結果が異なる.
150 WRITE A%;B!,C#;D$
160 END

```

13. データの編集出力

```

100 PRINT USING "!";"NEC COMPUTER"
110 PRINT USING "&"&"NEC COMPUTER"
120 PRINT USING "#####";123.456
130 PRINT USING "#####.##";123.456
140 PRINT USING "+#####.##";123.456
150 PRINT USING "#####.##-";123.456
160 PRINT USING "#####.##";123.456
170 PRINT USING "¥#####.##";123.456
180 PRINT USING "※#####.##";123.456
190 PRINT USING "#####.##";1234.56
200 PRINT USING "#####.##^";1234.56
210 PRINT USING "¥#####.##-";123.456
220 PRINT USING "#####";123456!
230 PRINT USING "NEC @ COMPUTER";"PERSONAL"
240 END

```

書式制御文字列の典型的な使い方。

14. 円の描画

```

100 SCREEN 0,0:CLS 3
110 FOR R=1 TO 100 STEP 5
120 CIRCLE(150,100),R,(R MOD 7)+1,,,R/100
130 NEXT R
140 ST=.00001
150 DATA 25,5.40,13,17
160 FOR I=1 TO 5
170 READ DAT:EN=ST+DAT/100*3.14*2
180 CIRCLE(450,100),100,I,-ST,-EN
190 ST=EN
200 NEXT I

```

半径、パレット番号、比率を変えながら楕円を描く。

DATA行(150行)の百分率データを円の角度に変換。
パレット番号、開始角度、終了角度を変えながら扇形を描き、円グラフを作る。

15. パレット(8色中・8色モード)

```

100 COLOR ,,,,0:CLS 3
110 FOR I=1 TO 7
120 COLOR=(I,7)
130 NEXT I
140 FOR I=1 TO 7
150 LINE(0,I*20)-STEP(639,10),I,BF
160 NEXT I
170 FOR I=1 TO 7
180 FOR J=1 TO 7
190 COLOR=(J,I)
200 GOSUB *WAITSUB
210 NEXT J
220 NEXT I
230 COLOR
240 END
250 *WAITSUB
260 FOR K=0 TO 100
270 NEXT K
280 RETURN

```

8色中・8色モード。

すべてのパレットを白に設定。

パレット番号ごとに1本ずつ、計7本の帯を描く。

各パレット番号の帯に、1~7のカラーコードを順に設定していく。

全パレットを初期化。

遅延用サブルーチン。

16. パレット(4096色中・16色モード)

```

100 SCREEN 3,0:COLOR ,,,,2:CLS 3 ———— 高分解能カラーモード, 4096色中・16色モード.
110 FOR I=1 TO 15
120   COLOR=(I,&HFFF) ———— すべてのパレットを白に設定.
130 NEXT I
140 FOR I=1 TO 15
150   LINE(0,I*20)-STEP(639,10),I,BF ———— パレット番号ごとに1本ずつ, 計15本の帯を描く.
160 NEXT I
170 FOR I=1 TO 15
180   FOR J=1 TO 15
190     COLOR=(J,16*I^2) ———— 各パレット番号の帯に, カラーコードを順に設定していく.
200     GOSUB *WAITSUB
210   NEXT J
220 NEXT I
230 COLOR ———— 全パレットを初期化.
240 END
250 *WAITSUB ———— 遅延用サブルーチン.
260 FOR K=0 TO 100
270 NEXT K
280 RETURN

```

17. DRAW

```

100 SCREEN 0,0,0,1:CLS 3
110 POINT(320,100)
120 AS="C4U60R60D60L60"
130 DRAW AS
140 DRAW "BE45A2S0.5X=AS;"
150 DRAW "A0BE10P"
160 PSET(440,40)
170 LINE-STEP(60,60),7,BF
180 PRESET(455,55)
190 LINE-STEP(30,30),0,B
200 PAINT(456,56),0
210 END

```

——— DRAWで図形を描く.

——— LINE, PAINTで同じ図形を描く.

18. グラフィックパターンの読み込みと表示

```

100 SCREEN 0,0:CLS 3:COLOR ,,,,1
110 XD=40:YD=20
120 BYTE=((XD+7)/8)*YD*3+4
130 FACT=BYTE*2+1
140 DIM G%(FACT)
150 LINE(0,0)-STEP(XD-1,YD-1),1,B
160 CIRCLE(XD/2-1,YD/2-1),YD/2,2
170 GET(0,0)-STEP(XD-1,YD-1),G% ———— 図形をG%()に読み込む.
180 FOR X=0 TO 500 STEP 100
190   PUT(X,100),G% ———— 読み込んだ図形を6か所にPUT.
200 NEXT

```

8色モードで40×20ドットのパターンを読み込むための配列変数G%()を確保する.

19. グラフィック画面のスクロール

```

100 SCREEN 1,0:CLS 3
110 FOR I=&H3000 TO &H4F00 STEP &H100
120   FOR J=&H21 TO &H7E
130     KCODE=I+J
140     PUT(X,168),KANJI(KCODE),PSET ———— 漢字をコード順に表示.
150     X=X+20
160     IF X>623 THEN X=0:ROLL 18 ———— 18ドット分ずつスクロールアップ.
170   NEXT J
180 NEXT I

```


20. ビューポートの設定

```

100 SCREEN 0,0:CLS 3
110 WINDOW(0,0)-(639,199) ———— ウィンドウの設定.
120 C=6:GOSUB *RECT
130 VIEW(1,1)-(638,99),0,7
140 C=5:GOSUB *RECT
150 VIEW(214,1)-(428,198),0,7
160 C=4:GOSUB *RECT
170 VIEW(0,0)-(639,199):CLS 3 ———— ビューポートをもとに戻す.
180 END
190 *RECT ———— べた塗りの長方形を描くサブルーチン.
200 LINE(50,50)-(600,150),C,BF
210 LOCATE 0,0:PRINT "どれかキーを押してください";
220 A$=INPUT$(1)
230 RETURN

```

21. ウィンドウの設定

```

100 SCREEN 0,0:CLS 3
110 C=1
120 VIEW(200,50)-(400,150),,7 ———— ビューポートの設定.
130 WINDOW(-100,-100)-(100,100)
140 GOSUB *CIRC
150 WINDOW(-100,-100)-(0,0)
160 GOSUB *CIRC
170 WINDOW(0,0)-(100,100)
180 GOSUB *CIRC
190 WINDOW(-500,-500)-(500,500)
200 GOSUB *CIRC
210 VIEW(0,0)-(639,199):CLS 3 ———— ビューポートをもとに戻す.
220 END
230 *CIRC ———— 円を描くサブルーチン.
240 CIRCLE(0,0),100,C
250 LOCATE 0,0:PRINT "どれかキーを押してください";
260 A$=INPUT$(1)
270 C=C+1:CLS 2
280 RETURN

```

22. エラー処理ルーチン

```

100 PRINT "XのY乗を求めます"
110 ON ERROR GOTO *ERRORMES ———— エラー発生時の分岐先を定義.
120 *START
130 INPUT "X:";X:INPUT "Y:";Y
140 IF X=0 THEN ERROR 250 ———— エラー番号250を独自に定義.
150 Z=X^Y:PRINT X;"の";Y;"乗は";Z;"です"
160 *RETRY
170 INPUT "もう一度やりますか(Y/N)";A$:PRINT
180 IF A$="Y" OR A$="y" THEN *START
190 ON ERROR GOTO 0 ———— プログラム終了時には必ず実行しておく.
200 END
210 *ERRORMES ———— エラー処理ルーチン.
220 IF ERR=250 THEN PRINT "定義されません" ———— エラー番号250のときの処理.
230 IF ERR=6 THEN PRINT "オーバーフローです" ———— オーバーフローのときの処理.
240 RESUME *RETRY ———— エラー処理ルーチンからの戻りにはRESUMEを使う.

```

23. 文字とキャラクタコードの変換

```

100 LINE INPUT "英数字を打ってください ";A$
110 IF A$="" THEN 190
120 B$=""
130 FOR I=1 TO LEN(A$)
140   D$=MID$(A$,I,1):D=ASC(D$)
150   IF D>=97 AND D<=122 THEN D$=CHR$(D-32) —— 英小文字のみを、英大文字に変換。
160   B$=B$+D$
170 NEXT I
180 PRINT:PRINT B$
190 END

```

24. 8進変換と16進変換

```

100 FOR I=0 TO 16
110   DE$=RIGHT$(" "+STR$(I),3)
120   OC$=RIGHT$(" "+OCT$(I),3)
130   HE$=RIGHT$(" "+HEX$(I),3) —— 0~16の整数を、10進表記、8進表記、16進表記の文字列に変換。
140   PRINT "10進:";DE$,
150   PRINT " 8進:";OC$,
160   PRINT "16進:";HE$
170 NEXT I

```

25. 文字列の抜き出し

```

100 INPUT"英数字をなにか打ってください ",A$
110 PRINT
120 PRINT "全体を2つに分割します"
130 AL$=LEFT$(A$,LEN(A$)/2) —— A$を左右半分ずつに分割する。
140 AR$=RIGHT$(A$,LEN(A$)-LEN(AL$)/2)
150 PRINT AL$;SPACE$(3);AR$
160 PRINT
170 PRINT "2文字ずつに分けます"
180 FOR I=1 TO LEN(A$) STEP 2 —— A$を2文字ずつに分ける。
190   PRINT MID$(A$,I,2);SPACE$(3);
200 NEXT I
210 END

```

26. 数値データの文字型化

```

100 OPEN "RDATA" AS #1 —— ランダムモードで"RDATA"をオープン。
110 FIELD #1,2 AS A$,4 AS B$,8 AS C$ —— 整数用2バイト、単精度実数用4バイト、倍精度実数用8バイトのフィールド変数を定義。
120 A%=987:A#=123.456:A#=#1234567891234#
130 LSET A$=MKI$(A%) —— 各型の数値を文字型データに変換し、
140 LSET B$=MKS$(A!) —— フィールド変数にセット。
150 LSET C$=MKD$(A#) ——
160 PUT #1 —— ファイルに書き込む。
170 PRINT "このプログラムで書き込むデータは"
180 PRINT "次のプログラムで読み出せます"
190 CLOSE:END

```

27. 文字型化数値データの数値化

```

100 OPEN "RDATA" AS #1 —— ランダムモードで"RDATA"をオープン。
110 FIELD #1,2 AS A$,4 AS B$,8 AS C$ —— 整数用2バイト、単精度実数用4バイト、倍精度実数用8バイトのフィールド変数を定義。
120 PRINT "前のプログラムで書き込んだデータを"
130 PRINT "読み出します"
140 GET #1 —— ファイルから読み込む。
150 A%=CVI(A$) ——
160 A!=CVS(B$) —— 文字型データを、各型の数値に戻す。
170 A#=#CVD(C$)
180 PRINT A%:PRINT A!:PRINT A#
190 CLOSE:END

```

28. シーケンシャルファイル

```

100 F$="DATA.D"+RIGHT$(DATE$,2)
110 OPEN F$ FOR OUTPUT AS #1 "DATA.Dxx" (xxは日付)を出力モードでオープン、
120 PRINT #1,DATE$;" ";TIME$ 日付と時刻を書き出す。
130 PRINT "今日のデータを入力してください"
140 PRINT:PRINT "リターンキーだけ押すと終了します"
150 PRINT:INPUT "品名";NA$
160 IF NA$="" THEN CLOSE:GOTO *INDATA 入力終了、ファイルのクローズ。
170 INPUT "価格";PRC
180 INPUT "個数";N%
190 PRINT #1,NA$;" ";PRC;" ";N% 品名、価格、個数の各データを書き出す。
200 GOTO 140
210 *INDATA
220 TOTAL=0
230 OPEN F$ FOR INPUT AS #1 同一ファイルを入力モードでオープン。
240 INPUT #1,DA$,TI$ 日付と時刻を読み込む。
250 PRINT:PRINT "日付      :      ";DA$,"時刻      :      ";TI$
260 PRINT
270 IF EOF(1) THEN *TOTALPRN
280 INPUT #1,NA$,PRC,N% 品名、価格、個数の各データを読み込む。
290 SUM=PRC*N% 小計。
300 PRINT NA$;TAB(10);PRC;"*";N%;"=";SUM 品名、価格、個数および小計の表示。
310 TOTAL=TOTAL+SUM 合計。
320 GOTO 270
330 *TOTALPRN
340 PRINT:PRINT "TOTAL=";TOTAL 合計の表示。
350 PRINT:PRINT "今日のデータを ";F$;" として作成しました"
360 CLOSE:END ファイルをクローズ、終了。

```

29. ランダムファイル

```

100 CLS:PRINT"住所録のファイル名は？"
110 PRINT:LINE INPUT F$
120 IF F$="" THEN 100
130 OPEN F$ AS #1 指定ファイル名のファイルをランダムモードでオープン。
140 FIELD #1,30 AS NAM$,20 AS TEL$,50 AS ADR$ 名前(30バイト), TEL (20バイト),住所(50バイト)の専用フィールド変数を定義。
150 *MENU
160 PRINT:PRINT
170 PRINT "<<メニュー>>":PRINT
180 PRINT "  検索／修正 : 1"
190 PRINT "  新しく登録 : 2"
200 PRINT "  終      わ      り : 0"
210 PRINT:PRINT "操作を番号で選択してください ";
220 Q$=INPUT$(1):PRINT Q$
230 ON VAL(Q$)+1 GOTO *EXIT,*INNAME,*NEWENTRY
240 GOTO *MENU
250 *INNAME 検索のルーチン。
260 IF LOF(1)<>0 THEN 330
270 PRINT:PRINT F$+" にはデータがありません"
280 PRINT:PRINT"新しく登録しますか？(Y/N) ";
290 Q$=INKEY$:IF Q$="" THEN GOTO 290
300 IF Q$="N" OR Q$="n" THEN GOTO *MENU
310 IF Q$="Y" OR Q$="y" THEN GOTO *NEWENTRY
320 GOTO 290
330 NH=1
340 PRINT:PRINT "探す名前は？":PRINT:INPUT "名前:";HNAM$ 検索したい名前の入力。
350 IF HNAM$="" THEN *MENU
360 HL=LEN(HNAM$)
370 IF NH>LOF(1) THEN PRINT:PRINT"その名前は登録されていません":GOTO *MENU
380 GET #1,NH
390 IF HNAM$<>LEFT$(NAM$,HL) THEN NH=NH+1:GOTO 370
400 PRINT:PRINT"名前:";NAM$
410 PRINT"TEL:";TEL$ 検索結果の表示。
420 PRINT"住所:";ADR$
430 PRINT:PRINT"修正しますか？(Y/N) ";
440 Q$=INKEY$:IF Q$="" THEN GOTO 440
450 IF Q$="N" OR Q$="n" THEN PRINT "N":GOTO *MENU
460 IF Q$="Y" OR Q$="y" THEN PRINT "Y":GOTO *CHANGE
470 GOTO 440

```

ファイルにまったくデータがない場合の例外処理。

ファイル中から1レコードずつ読み込み、名前を検索する。

修正の要、不要を決める。

```

480 *CHANGE _____ 修正のルーチン。
490 PRINT:PRINT"リターンキーだけ押すとその項目は修正されません"
500 DNAM$="";DTEL$="";DADR$=" "
510 PRINT:INPUT"名前";DNAM$
520 IF DNAM$<>" " THEN LSET NAM$=DNAM$
530 INPUT"TEL";DTEL$
540 IF DTEL$<>" " THEN LSET TEL$=DTEL$
550 INPUT"住所";DADR$
560 IF DADR$<>" " THEN LSET ADR$=DADR$
570 PUT #1,NH:GOTO *MENU _____ 修正後のデータを書き出す。
580 *NEWENTRY _____ 新規登録のルーチン。
590 PRINT:PRINT"新しく登録します"
600 NH=LOF(1)+1 _____ ファイルの最終レコードの次のレコードを指定。
610 PRINT:INPUT"名前";NNAM$
620 IF NNAM$="" THEN *MENU
630 INPUT"TEL";NTEL$
640 INPUT"住所";NADR$
650 LSET NAM$=NNAM$
660 LSET TEL$=NTEL$
670 LSET ADR$=NADR$
680 PUT #1,NH:GOTO *MENU _____ データを書き出す。
690 *EXIT _____ 終了のルーチン。
700 PRINT:PRINT"終了"
710 CLOSE:END _____ ファイルのクローズ、終了。

```

各データを各フィールド変数にセット。

各データを各フィールド変数にセット。

30. キー割り込みルーチン

```

100 CLS
110 ON KEY GOSUB *SINSUB,*COSSUB,*ENDSUB _____ f・1, f・2, f・3が押されたときの分岐先を定義。
120 FOR I=1 TO 3
130 KEY(I) ON _____ f・1, f・2, f・3の各キーの割り込みを許可。
140 NEXT I
150 DEG=0:PI=3.1416:FLAG=1
160 PRINT "f・1 または f・2 を押してください (f・3 で終了). "
170 *START
180 TH=DEG/180*PI
190 LOCATE 0,10
200 ON FLAG GOTO *SINC,*COSC
210 *SINC:PRINT "TH=";DEG;TAB(12);"SIN(TH)=";SIN(TH);:GOTO *EXIT _____ 10度ごとの
220 *COSC:PRINT "TH=";DEG;TAB(35);"COS(TH)=";COS(TH);:GOTO *EXIT _____ 正弦値を表示。
230 *EXIT:DEG=DEG+10 _____ 10度ごとの
240 FOR I=0 TO 1000:NEXT _____ 表示遅延用タイマー。
250 LOCATE 0,10:PRINT STRING$(70," ");
260 GOTO *START
270 *SINSUB:FLAG=1:RETURN _____ f・1が押されたらFLAG=1にする。
280 *COSSUB:FLAG=2:RETURN _____ f・2が押されたらFLAG=2にする。
290 *ENDSUB:KEY OFF:END _____ f・3が押されたら終了。

```

31. タイマー割り込みルーチン

```

100 CLS:CONSOLE ,,,1:FLG=0
110 PRINT "目覚まし時計"
120 LOCATE 0,10:PRINT "現在の時刻は ";TIMES$
130 LOCATE 0,2:INPUT "セット時刻は (HH:MM:SS) ";ST$
140 ON TIMES$=ST$ GOSUB 200 _____ セット時刻到来時の分岐先を定義。
150 TIMES$ ON _____ クロック割り込みを許可。
160 LOCATE 13,10:PRINT TIMES$; _____ 現在時刻の表示。セット時刻までは
170 IF FLG=0 THEN 160 _____ 160行～170行が繰り返し実行される。
180 CLS:LOCATE 30,10:PRINT "おはよう！！！！"
190 END
200 PRINT
210 COLOR 7:BEEP 1
220 PRINT "もう起きる時間ですよ！！！！";
230 IF INKEY$="" THEN 220 _____ なにかキーを押さない限り、メッセージが表示され、スピーカは鳴りっぱなしとなる。
240 BEEP 0:FLG=1 _____ なにかキーが押されると、スピーカは鳴りやみ、メインルーチンに戻る。
250 RETURN

```


32. 乱数の発生

```

100 RANDOMIZE
110 DIM SUM(6)
120 FOR I=1 TO 100
130   DA=INT(RND*6+1)
140   SUM(DA)=SUM(DA)+1
150 NEXT I
160 FOR I=1 TO 6
170   PRINT I;"-";SUM(I);"%"
180 NEXT I

```

33. テキスト画面のモード設定と文字に対する色・機能の設定

```

100 WIDTH 80,25:CLS
110 CONSOLE 0,25,0,0 ————— 白黒モード.
120 LOCATE 0,0
130 PRINT "白黒モード":PRINT
140 GOSUB *PRINTCHR
150 GOSUB *KEYSUB
160 CONSOLE 0,25,0,1 ————— カラーモード.
170 LOCATE 0,0
180 PRINT "カラーモード":PRINT
190 GOSUB *PRINTCHR
200 END
210 *PRINTCHR
220 FOR I=0 TO 31
230   PRINT "-COMPUTER-";
240 NEXT I
250 GOSUB *KEYSUB
260 FOR I=0 TO 7
270   COLOR@(I*10,2)-(I*10+9,5),I ——— 文字にファンクションコードを設定.
280 NEXT I
290 RETURN
300 *KEYSUB ————— キー入力待ちサブルーチン.
310 LOCATE 0,10
320 PRINT "どれかキーを押してください";
330 A$=INPUT$(1)
340 RETURN

```

34. 利用者定義文字の登録

```

100 DIM CHRPTN%(17) ————— 利用者定義文字パターンを格納するための配列変数を確保.
110 CHRPTN%(0)=16:CHRPTN%(1)=16
120 FOR I=2 TO 17:CHRPTN%(I)=2^(I-2)-1:NEXT I ——— パターンを格納する.
130 KPLOAD &H762A,CHRPTN% ————— パターンをシステムに登録.
140 FOR I=0 TO 4
150   PRINT KNJ$("1B4B")+KNJ$("762A")+KNJ$("1B48"); ——— 表示.
160 NEXT I

```

35. 2バイト系日本語文字列の変換と抜き出し

```

100 A0$="いろは漢字ABC"
110 PRINT A0$
120 K0$=AKCNV$(A0$) ————— 1バイト系英数カナ文字を2バイト系全角文字に変換.
130 PRINT K0$
140 K1$=KMID$(K0$,1,4)+KMID$(K0$,7,4) ——— 2バイト系全角文字に変換された文字列から
150 PRINT K1$ ——— "漢字"を取り除く.
160 A1$=KACNV$(K1$) ————— 2バイト系全角文字を1バイト系英数カナ文字に変換.
170 PRINT A1$

```

36. 2バイト系日本語文字列と漢字コードの相互変換

```

100 A$=JIS$(KMID$("漢字",2,1)) —— 最初のKIコードをスキップし、
110 PRINT A$ —— "漢"のJISコードを求める。
120 B$=KNJ$("1B4B")+KNJ$(A$)+KNJ$("1B48") —— "漢"を表示するため、前後にKI
130 PRINT B$ —— /KOコードを付加する。

```

37. 特定タイプの文字列の抜き出し

```

100 A$="イロハ漢字ABC"
110 PRINT A$
120 KI$=CHR$(&H1B)+CHR$(&H4B):KO$=CHR$(&H1B)+CHR$(&H48)
130 PRINT KEXT$(A$,0) —— 1バイト系英数カナ文字のみを抜き出して表示。
140 PRINT KI$+KEXT$(A$,1)+KO$ —— 2バイト系日本語文字を抜き出し、前後
—— にKI/KOコードを付加して表示。

```

38. 2バイト系日本語文字を含む文字列の長さタイプ

```

100 A$="ABC漢字イロハ"
110 FOR I=0 TO 5
120 PRINT KLEN(A$,I): —— 各タイプの文字の合計数を、タイプ別に表示。
130 NEXT I
140 PRINT:PRINT
150 FOR J=1 TO KLEN(A$,0) —— 文字列中の各文字のタイプを順に表示。
160 PRINT KTYPE(A$,J);
170 NEXT J

```

39. 利用者定義関数

```

100 DEF FNMENSEKI(R)=3.14159*R*R —— 円の面積を求める利用者定義関数の定義。
110 DEF FNENSHUU(R)=3.14159*2*R —— 円周を求める利用者定義関数の定義。
120 INPUT "半径は";HANKEI
130 PRINT "面積:";FNMENSEKI(HANKEI)
140 PRINT "円周:";FNENSHUU(HANKEI)
150 END

```

付 録

A. エラーメッセージとその対策

BASIC の実行中にエラーが発生すると、メッセージが表示されます。ここでは、こうしたエラーメッセージと、その原因および対策について説明します。

メッセージの右側の数字(69 など)は、“エラーコード”と呼ばれるもので、エラー発生時, ERR (関数)に格納されます。

“原因:”には、そのエラーが発生した際の原因を簡略に説明してあります。複数の原因が考えられる場合は、列挙してあります。

“対策:”には、そのエラーの原因をとり除く方法か、またはエラーの発生を未然に防ぐための対策が説明してあります。

エラーについての詳細は、第 3 章中の ERL/ERR, ERROR, ON ERROR GOTO, RESUME などの項をご覧ください。

Bad allocation table	69
----------------------	----

原因: ディスク内部の FAT が壊れている。

対策: データを読み取ることができないので、ディスクを再フォーマットする。ただし、以前のファイルはすべて失われる。

Bad drive number	70
------------------	----

原因: LOAD, SAVE, KILL, OPEN などのファイル操作命令でドライブの指定がまちがっている。たとえばシステムにつながっていないドライブを指定したなど。

対策: 正しいドライブを指定する。

Bad file name	56
---------------	----

原因: LOAD, SAVE, KILL, OPEN などのファイル操作命令で、ファイル名の指定がまちがっている。

対策: 正しいファイル名を指定する。

Bad file number	52
-----------------	----

原因: 同時オープンファイル数を超えるファイル番号を指定しようとした。

対策: ファイル番号を小さくするか、同時オープンファイル数を増やす。

Bad track / sector	71
--------------------	----

原因: DSKO\$, DSKI\$ で、ヘッド番号、トラック、セクタの指定が誤っている。

対策: 正しいヘッド番号、トラック、セクタを指定する。

Can't continue

17

原因 : STOP 命令や CTRL+C などを実行を停止した場合、プログラムを書き換えたりすると、CONT によって実行を再開することができない。

対策 : CONT を使用したい場合は、実行の中断中にプログラムを書き換えてはならない。実行中断のタイミングによっては、プログラムを書き換えたりしなくとも実行の再開ができなくなることもあるので注意。

Direct statement in file

57

原因 : LOAD でアスキー形式のプログラムファイルをロードするときに、ダイレクトステートメント（行番号のない行）が存在した。

対策 : プログラムファイルの行番号が何かの理由で破壊されているか、あるいは、指定ファイルがプログラムファイルではないデータファイルであると考えられるので、ファイルの内容を確認する。

Disk full

68

原因 : SAVE, PRINT #, PUT などのディスクに書き出す命令を実行したとき、ディスク上に空き領域がなかった。

対策 : 不要なファイルを削除して、空き領域を確保する。

別のドライブ上のディスクを使用する。

同じドライブ上でディスクを差し替えて使用する。ただしこの方法は、ファイルがオープン状態のときには使えないので注意。

Disk I / O error

64

原因 : ディスクとの入出力中にエラーが発生した。

対策 : 入出力を行おうとしたディスクがフォーマットされていないか、または物理的に破壊されていると考えられるので、ディスクの再フォーマットを行ってみる。

Disk offline

62

原因 : LOAD, SAVE, KILL, OPEN などのファイル操作命令で、ディスクのセットされていないドライブに対して入出力を行おうとした。

対策 : ディスクを正しくドライブにセットする。

Division by Zero (/0)

11

原因 : 0 による除算、すなわち $n / 0$ (n は任意の数) や 0^{-1} が実行されて、その結果得られた値がオーバーフローを起こした。

対策 : 除数が 0 にならないようにする。

Duplicate Difinition

10

原因：一度宣言した配列を二重定義しようとした。

DIM で宣言しないで配列を使った場合に、その配列を再定義しようとした。

対策：別の配列名を使うか、あるいは ERASE で古い配列を消去した上で再び同じ配列名で定義しなおす。

Duplicate label

31

原因：プログラム中に同じラベル名が2つ以上存在している。

対策：ラベル名を変更して同一名のラベル名が存在しないようにする。

Feature not available

33

原因：現在の状態では使用できない命令を実行しようとした。

対策：その命令を実行するために必要なハードウェアなどを装備あるいは準備する。

FIELD overflow

50

原因：FIELD で、ランダムファイルのレコード長として合計 256 バイト以上の領域を指定した。

対策：フィールドの合計長は 255 バイト以内にする。

File already exist

65

原因：NAME によって新たに付けようとしたファイル名がすでに存在している。

対策：新しく付けるファイル名を変更するか、すでに存在しているファイルのファイル名を変更する。

File already open

54

原因：すでにオープンされているファイルに対して、OPEN, KILL, NAME などを行った。

対策：CLOSE でいったんファイルを閉じる。

File not found

53

原因：LOAD, SAVE, KILL, NAME などのファイル操作命令で、指定したファイルが見つからない。

対策：正しいドライブ名およびファイル名を指定する。

File not open

60

原因：PRINT #, INPUT #など、ファイル番号を指定して入出力を行う命令で、オープンされていないファイルを参照しようとした。

対策：入出力命令を実行する前に、OPEN でファイルをオープンしておく。

File write protected

61

原因：次のうちいずれかの書き込み禁止属性が付けられているファイルに、書き込みを行おうとした。

- ・SET によって、ファイルディスクリプタ、ドライブのディスク、ファイル番号に付けられたもの。
- ・フロッピーディスクに張られたシールによるもの。

対策：書き込み禁止を解除する。

FOR without NEXT

26

原因：FOR～NEXT が正しく対応していない。

対策：プログラムの流れをたどり、FOR～NEXT を正しく対応させる。

Illegal direct

12

原因：ダイレクトモードで使用できない命令をダイレクトモードで使った。

対策：プログラムモード上で実行する。

Illegal function call

5

原因：命令や関数の使い方がまちがっている。すなわち、引数とその関数の許容する範囲を超えたり、結果がその関数のとり得る範囲を超えたりしている。たとえば、次のような場合である。

- ・LOG 関数で負や 0 の引数を指定した。
- ・SQR 関数で負の引数を指定した。
- ・MID\$, LEFT\$, RIGHT\$, STRING\$, SPACE\$, INSTR, ON…GOSUB などにおいて、不適当な引数が用いられた。
- ・ほか。

対策：原因がさまざま考えられるので、各命令、関数の使い方をリファレンスで確認する。

Illegal operation

74

原因：スクロールウィンドウ内に表示データがいっぱい詰まっている場合、またはスクロールウィンドウ内でキー入力したデータがいっぱい詰まっている場合に、スクロール開始行に対してインサートモードでキー入力した。

対策：インサートモードを抜けてから、キー入力を行う。

Input past end

55

原因：INPUT #, GET などのファイルからデータを読み込む命令で、ファイル中のすべてのデータを読み尽くしたあとに、さらに入力命令が実行された。

対策：ファイル中のデータの数と、入力命令で読み込む数とを合わせる。

EOF, LOF などの関数を使って、ファイルの終了を検出するようにする。

Line buffer overflow

23

原因：1 行で入力できる文字の範囲（255 バイト）を超えて入力が行われた。

対策：1 行の文字数を 255 バイト以内にする。

Missing operand

22

原因：命令中で、必要なパラメータが指定されていない。

対策：書式を確かめ、必要なパラメータを指定する。

NEXT without FOR

1

原因：FOR～NEXT が正しく対応していない。

対策：プログラムの流れをたどり、FOR～NEXT を正しく対応させる。

No RESUME

19

原因：エラー処理ルーチンの終わりに RESUME がなく、プログラムの実行が継続できない。

対策：RESUME, END, ON ERROR GOTO 0 のどれかで終わらせるようにする。

Out of DATA

4

原因：DATA 行中のデータが足りず、READ で読むべきデータがない。

対策：DATA 行中のデータの数と、READ で読み込む数とを合わせる。READ と DATA の対応と、RESTORE が正しく使われているかどうかを確認する。

Out of memory

7

原因：次のような理由により、メモリ容量が足りなくなった。

- ・プログラムが長すぎる。
- ・配列が大きすぎる。
- ・FOR～NEXT, GOSUB などネスティング(入れ子構造)が深くなりすぎて、スタックがいっぱいになった。
- ・ほか。

対策：プログラム、配列を小さくする。ネスティングを浅くする。

FRE を使ってフリーなメモリを調べてみて、メモリの空き容量が異常に少ない場合には、そのプログラムを実行する前に別のプログラム中で、BASIC のプログラム領域を小さく設定したままにしている可能性がある。このようなときには、CLEAR を実行して、BASIC の使うメモリを大きくする。また、起動時にファイルバッファの容量を多く取っていることもあるので、再起動して同時オープンファイル数を必要最小限にする。

Out of string space

14

原因：文字列の全体量が、使用できるメモリ容量を超えてしまった。

対策：文字列、配列文字列を小さくする。

Overflow (OV)

6

原因：演算結果や入力された数値が、許される範囲を超えた。

対策：データの型と、とる値の範囲が正しいかどうかを確認する。

Rename across disks

73

原因：異なるドライブ上にあるファイルに対して NAME を実行しようとした。

対策：NAME では同じドライブ上のファイルを指定する。

RESUME without error

20

原因：エラー処理ルーチンでないところに、RESUME がある。

対策：ON ERROR GOTO で指定されたところ以外、たとえばメインルーチン内に RESUME があるとこのエラーが発生するので、メインルーチンの終わりには必ず END を置くようにする。プログラムの流れを確認すること。

RETURN without GOSUB

3

原因：GOSUB～RETURN が正しく対応していない。

対策：GOSUB で呼び出されたサブルーチン以外、たとえばメインルーチン内に RETURN があるとこのエラーが発生するので、プログラムの流れに沿って、GOSUB と RETURN の対応を確認する。

Sequential I/O only

59

原因：シーケンシャル入出力以外の入出力命令を使用した。

MERGE でバイナリファイルを指定した。

対策：シーケンシャル入出力命令を使用するようにする。

MERGE でディスクから読み込むファイルは、アスキー形式でセーブしておく。

String formula too complex

16

原因：文字式が複雑すぎる。

対策：複雑な文字式は、いくつかの式に分ける。

String too long

15

原因：1つの文字変数内の文字数が 255 バイトを超えている。

対策：2つの変数に分けて処理する。

Subscript out of range

9

原因：配列変数の添字の値が DIM および OPTION BASE で指定されたときの範囲を外れている。

DIM による宣言を省略した場合に、添字の値が 10 を超えている。

対策：配列変数の定義による添字の範囲と引用する配列変数の範囲を確認する。

Syntax error 2

原因：命令が書式どおりになっていない。

予約語以外の命令がある。

READ と DATA との対応で、読み込む変数とデータとの間で型の不一致が生じた。

対策：各命令を書式どおり正しく記述する。

READ と DATA では、読み込む変数とデータの型を合わせる。

Tape read error 27

原因：テープからの読み込み時にエラーが発生した。

対策：再度やってみる。データレコーダ、テープの異常を調べる。

Type mismatch 13

原因：式の左辺、右辺、関数の引数などにおいて、変数の型が一致していない。

対策：関数に与えるデータの型を確認する。

Undefined label 32

原因：定義されていないラベル名を参照した。

対策：ラベル名を正しく定義する。

Undefined line number 8

原因：GOTO, GOSUB, IF...THEN~ELSEなどで、存在しない行番号を飛び先に指定した。

対策：行番号を確認する。

Undefined user function 18

原因：定義されていない利用者定義関数、または機械語関数を引用した。

対策：利用者定義関数は、引用する前に DEF FN で定義する。機械語関数は、DEF USR で実行開始アドレスを定義しておく。

Unprintable error 21, 24, 25, 28, 他

原因：メッセージの定義されていないエラー。

対策：ERROR 〈整数表記〉を実行してこのエラーが生ずる場合、その〈整数表記〉番のエラーを利用者独自のエラー処理に利用することができる。

WEND without WHILE 30

原因：WHILE~WEND が正しく対応していない。

対策：プログラムの流れをたどり、WHILE~WEND を正しく対応させる。

WHILE without WEND

29

原因 : WHILE～WEND が正しく対応していない。

対策 : プログラムの流れをたどり、WHILE～WEND を正しく対応させる。

システム予約

51

B. コントロールコード表

キャラクタ コード	対応する キー	エディタ中の機能	プログラム中の機能
1	CTRL+A	HELP キーと同じ	—
2	CTRL+B	カーソルを1つ前のワードへ移動	—
3	CTRL+C	STOP キーと同じ	—
4	CTRL+D	1 ワード削除	—
5	CTRL+E	カーソル位置から行末までを消去	—
6	CTRL+F	カーソルを1つ先のワードへ移動	—
7	CTRL+G	スピーカを鳴らす	スピーカを鳴らす
8	CTRL+H	BS キーと同じ	バックスペース
9	CTRL+I	TAB キーと同じ	水平タブ
10	CTRL+J	行の連結(インサートモードでは行の分割)	ラインフィード
11	CTRL+K	カーソルをホームポジションへ移動	ホームポジション
12	CTRL+L	テキスト画面のクリア	テキスト画面のクリア
13	CTRL+M	リターンキーと同じ	キャリッジリターン
15	CTRL+O	画面への表示の無効／有効の切り換え	—
18	CTRL+R	インサートモードに入る／出る	—
19	CTRL+S	表示(スクロール)の一時停止	—
21	CTRL+U	1 行消去	—
24	CTRL+X	カーソルを1行の最後へ移動	—
28	→	カーソルを右へ移動	カーソルを右へ移動
29	←	カーソルを左へ移動	カーソルを左へ移動
30	↑	カーソルを上へ移動	カーソルを上へ移動
31	↓	カーソルを下へ移動	カーソルを下へ移動

備考 プログラム中の機能は、対応するコードが画面に出力される際(プログラムの実行中)に働きます。

C. キャラクタコード表

上位 4 ビット →

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
下位 4 ビット ↓	0	^{D_E}		0	@	P	'	p			ー	タ	ミ			×
	1	^{S_H}	^{D₁}	!	1	A	Q	a	q		。	ア	チ	ム		円
	2	^{S_X}	^{D₂}	"	2	B	R	b	r		「	イ	ツ	メ		年
	3	^{E_X}	^{D₃}	#	3	C	S	c	s		」	ウ	テ	モ		月
	4	^{E_T}	^{D₄}	\$	4	D	T	d	t		、	エ	ト	ヤ		日
	5	^{E_Q}	^{N_K}	%	5	E	U	e	u		・	オ	ナ	ユ		時
	6	^{A_K}	^{S_N}	&	6	F	V	f	v		ヲ	カ	ニ	ヨ		分
	7	^{B_L}	^{E_B}	'	7	G	W	g	w		ア	キ	ヌ	ラ		秒
	8	^{B_S}	^{C_N}	(8	H	X	h	x		イ	ク	ネ	リ	♠	
	9	^{H_T}	^{E_M})	9	I	Y	i	y		ウ	ケ	ノ	ル	♥	
	A	^{L_F}	^{S_B}	*	:	J	Z	j	z		エ	コ	ハ	レ	♦	
	B	^{H_M}	^{E_C}	+	;	K	[k	{		オ	サ	ヒ	ロ	♣	
	C	^{C_L}	→	,	<	L	¥	l			ヤ	シ	フ	ワ	●	
	D	^{C_R}	←	—	=	M]	m	}		ユ	ス	ヘ	ン	○	
	E	^{S_O}	↑	.	>	N	^	n	~		ヨ	セ	ホ	〃		
	F	^{S_I}	↓	/	?	O	_	o			ツ	ソ	マ	°		

D. 誘導関数

関数として用意していない三角関数のうち、いくつかは、用意された関数を使って作り出すことができます。以下の数式を参考にしてください(誤差の範囲に注意が必要です)。

目的とする関数	組み込み関数からの誘導式
セカント	$\text{SEC}(X) = 1/\text{COS}(X)$
コセカント	$\text{CSC}(X) = 1/\text{SIN}(X)$
コタンジェント	$\text{COT}(X) = 1/\text{TAN}(X)$
アークサイン	$\text{ARCSIN}(X) = \text{ATN}(X/\text{SQR}(-X * X + 1))$
アークコサイン	$\text{ARCCOS}(X) = -\text{ATN}(X/\text{SQR}(-X * X + 1)) + 1.5708$
アークセカント	$\text{ARCSEC}(X) = \text{ATN}(\text{SQR}(X * X - 1)) + (\text{SGN}(X) - 1) * 1.5708$
アークコセカント	$\text{ARCCSC}(X) = \text{ATN}(1/\text{SQR}(X * X - 1)) + (\text{SGN}(X) - 1) * 1.5708$
アークコタンジェント	$\text{ARCCOT}(X) = -\text{ATN}(X) + 1.5708$
ハイパーボリック・サイン	$\text{SINH}(X) = (\text{EXP}(X) - \text{EXP}(-X))/2$
ハイパーボリック・コサイン	$\text{COSH}(X) = (\text{EXP}(X) + \text{EXP}(-X))/2$
ハイパーボリック・タンジェント	$\text{TANH}(X) = -\text{EXP}(-X)/(\text{EXP}(X) + \text{EXP}(-X)) * 2 + 1$
ハイパーボリック・セカント	$\text{SECH}(X) = 2/(\text{EXP}(X) + \text{EXP}(-X))$
ハイパーボリック・コセカント	$\text{CSCH}(X) = 2/(\text{EXP}(X) - \text{EXP}(-X))$
ハイパーボリック・コタンジェント	$\text{COTH}(X) = \text{EXP}(-X)/(\text{EXP}(X) - \text{EXP}(-X)) * 2 + 1$
ハイパーボリック・アークサイン	$\text{ARCSINH}(X) = \text{LOG}(X + \text{SQR}(X * X + 1))$
ハイパーボリック・アークコサイン	$\text{ARCCOSH}(X) = \text{LOG}(X + \text{SQR}(X * X - 1))$
ハイパーボリック・アークタンジェント	$\text{ARCTANH}(X) = \text{LOG}((1 + X)/(1 - X))/2$
ハイパーボリック・アークセカント	$\text{ARCSECH}(X) = \text{LOG}((\text{SQR}(-X * X + 1) + 1)/X)$
ハイパーボリック・アークコセカント	$\text{ARCCSCH}(X) = \text{LOG}((\text{SGN}(X) * \text{SQR}(X * X + 1) + 1)/X)$
ハイパーボリック・アークコタンジェント	$\text{ARCCOTH}(X) = \text{LOG}((X + 1)/(X - 1))/2$

E. 予約語一覧

これらの予約語を、変数名やラベル名として使用することはできません。ただし、予約語を含む文字列（例：ABSN）は、使用してもかまいません。

ABS	CONT	EOF	INPUT
AKCNV\$	COPY	EQV	INPUT\$
ALLOC	COS	ERASE	INSTR
AND	CSNG	ERL	INT
ASC	CSRLIN	ERR	IRESET
ATN	CVD	ERROR	ISSET
ATTR\$	CVI	EXP	JIS\$
AUTO	CVS	FIELD	KACNV\$
BEEP	DATA	FILES	KANJI
BLOAD	DATE\$	FIX	KEXT\$
BSAVE	DEF	FN	KEY
CALL	DEFDBL	FOR	KILL
CDBL	DEFINT	FRE	KINPUT
CHILD	DEFSNG	GET	KINSTR
CHR\$	DEFSTR	GOSUB	KLEN
CINT	DELETE	GOTO	KMID\$
CIRCLE	DELIM	GO TO	KNJ\$
CLEAR	DIM	HELP	KPLOAD
CLOSE	DRAW	HEX\$	KTYPE
CLS	DSKF	IEEE	LEFT\$
CMD	DSKI\$	IF	LEN
COLOR	DSKO\$	IFC	LET
COM	EDIT	IMP	LFILES
COMMON	ELSE	INKEY\$	LINE
CONSOLE	END	INP	LIST

LLIST	ON	RESTORE	TAB
LOAD	OPEN	RESUME	TAN
LOC	OPTION	RETURN	THEN
LOCATE	OR	RIGHT\$	TIMES
LOF	OUT	RND	TIMEOUT
LOG	PAINT	ROLL	TO
LPOS	PEEK	RSET	TROFF
LPRINT	PEN	RUN	TRON
LSET	PLAY	SAVE	USING
MAIL	POINT	SCREEN	USR
MAP	POKE	SEARCH	VAL
MERGE	POLL	SEG	VARPTR
MID\$	POS	SET	VIEW
MKD\$	PPOLL	SGN	WAIT
MKI\$	PPR	SIN	WBYTE
MKS\$	PRESET	SPACE\$	WEND
MOD	PRINT	SPC	WHILE
MON	PSET .	SQR	WIDTH
MOTOR	PUT	SRQ	WINDOW
NAME	RANDOMIZE	STATUS	WRITE
NEXT	RBYTE	STEP	XOR
NEW	READ	STOP	
NOT	REM	STR\$	
OCT\$	REN	STRING\$	
OFF	RENUM	SWAP	

索引

A

ABS	38
AKCNV\$	38
AND	22
ASC	38
ATN	39
ATTR\$	39
AUTO	40

B

BEEP	40
BLOAD	40
BSAVE	41

C

CALL	42
CDBL	42
CHAIN	43
CHR\$	44
CINT	44
CIRCLE	45
CLEAR	46
CLOSE	47
CLS	47
COLOR	48,51
COLOR@	52
COMMON	52
COM ON/OFF/STOP	53
CONSOLE	54
CONT	55
COPY	55
COS	56
CSNG	57

CSRLIN	57
CVD	58
CVI	58
CVS	58

D

DATA	58
DATE\$	59
DEF FN	59
DEFDBL	60
DEFINT	60
DEFSEG	60
DEFSTR	60
DEF SEG	61
DEF USR	61
DELETE	62
DIM	62
DISK モード	3
DRAW	63
DSKF	67
DSKI\$	68
DSKO\$	68

E

EDIT	69
END	70
EOF	70
EQV	22
ERASE	70
ERL	71
ERR	71
ERROR	71
EXP	72

F

FIELD	72
FILES	73
FIX	74
FOR...TO...STEP~NEXT	74
FPOS	75
FRE	75

G

GET	76
GET@	77
GOSUB	78
GOTO	78
GO TO	78

H

HELP ON/OFF/STOP	79
HEX\$	80

I

IF...GOTO~ELSE	80
IF...THEN~ELSE	80
IMP	22
INKEY\$	81
INP	81
INPUT	82
INPUT #	83
INPUT\$	83
INPUT WAIT	84
INSTR	84
INT	85

J

JIS\$	85
-------------	----

K

KACNV\$	85
KEXT\$	86
KEY	86

KEY LIST	87
KEY ON/OFF/STOP	87
KILL	88
KINPUT	88
KINSTR	89
KLEN	89
KMID\$	90
KNJ\$	90
KPLOAD	91
KTYPE	92

L

LEFT\$	92
LEN	93
LET	93
LFILES	73
LINE	94
LINE INPUT	95
LINE INPUT WAIT	96
LINE INPUT #	96
LIST	97
LLIST	97
LOAD	97
LOAD ?	98
LOC	98
LOCATE	99
LOF	99
LOG	100
LPOS	100
LPRINT	100
LPRINT USING	101
LSET	101

M

MAP	102
MERGE	103
MID\$	103,104
MKD\$	104
MKI\$	104
MKS\$	104

MOD	20
MON	105
MOTOR	105

N

NAME	106
NEW	106
NEW ON	106
NOT	21

O

OCT\$	107
ON COM GOSUB	108
ON ERROR GOTO	109
ON...GOSUB	109
ON...GOTO	109
ON HELP GOSUB	110
ON KEY GOSUB	111
ON PEN GOSUB	112
ON STOP GOSUB	112
ON TIME\$ GOSUB	113
OPEN	114
OPTION BASE	116
OR	22
OUT	117
OV	21,186

P

PAINT	117,118
PEEK	120
PEN	120
PEN ON/OFF/STOP	121
POINT	122,123
POKE	123
POS	124
PRESET	124
PRINT	125
PRINT #	126
PRINT USING	128
PRINT # USING	130

PSET	130
PUT	131
PUT @	132

R

RANDOMIZE	133
READ	134
REM	134
RENUM	135
RESTORE	135
RESUME	136
RETURN	136
RIGHT\$	137
RND	137
ROLL	138
ROM モード	3
RSET	101
RUN	138

S

SAVE	139
SCREEN	140
SEARCH	143
SET	144
SGN	145
SIN	145
SPACE\$	146
SPC	146
SQR	146
STOP	147
STOP ON/OFF/STOP	147
STR\$	148
STRING\$	148
SWAP	149

T

TAB	149
TAN	149
TERM	150
TIME\$	152

TIMES\$ ON/OFF/STOP	152
TROFF	153
TRON	153

U

USR	153
-----------	-----

V

VAL	154
VARPTR	154
VIEW	155,156

W

WAIT	157
WHILE~WEND	158
WIDTH	158
WIDTH LPRINT	159
WINDOW	160,161
WRITE	161
WRITE #	162

X

XOR	22
-----------	----

あ

エラーメッセージ	30,181
演算子	19
演算の優先順位	25
オーバーフロー	20

か

拡張ファイル名	27
型宣言文字	15
型変換	17
関係演算	21
関数	24
キャラクタコード	190
行	9
行番号	9
コントロールコード	189

さ

算術演算	19
式	19
次元	16
指数	19
実数型定数	13
数値型定数	12
整数型定数	12
セーブ	6
添字	15

た

単精度実数型定数	13
ダイレクトモード	4
定数	12
デバイス名	26
デフォルト値	35

は

倍精度実数型定数	14
配列変数	15
パラメータ	36
ファイル	25
ファイルディスクリプタ	26
ファイル番号	27
ファイル名	27
プログラムモード	5
プロンプト	4
文	9
変数	14
変数の型	14
変数名	14

ま

マルチステートメント	9
文字型定数	12
文字列の演算	24

や

予約語	17,192
誘導関数	191

ら

ラベル名	29
ロード	6
論理演算	21

わ

割り込み	28
------	----

記号

8 進形式	13
10 進形式	12
16 進形式	13
(スペース)	11
, (コンマ)	11
. (ピリオド)	11
: (コロソ)	11
; (セミコロソ)	11
? (クエツション)	11
! (エクスクラメツション)	15
^ (アップアロー)	19
/ (スラッシュ)	19
' (アポストロフイ)	11
- (マイナス)	11,19
十 (プラス)	19
= (イコール)	21
< (小なり)	21
> (大なり)	21
¥ (エン)	20
\$ (ドル)	15
% (パーセント)	15
# (シャープ)	15
* (アスタリスク)	11,19

